A GENERIC ARCHITECTURE FOR INTERACTIVE INTELLIGENT TUTORING SYSTEMS

A Dissertation submitted for the Degree of Doctor of Philosophy

Tajudeen Abayomi Atolagbe

Department of Information Systems and Computing

Brunel University

April, 2001

ABSTRACT

This research is focused on developing a generic intelligent architecture for an interactive tutoring system.

A review of the literature in the areas of instructional theories, cognitive and social views of learning, intelligent tutoring systems development methodologies, and knowledge representation methods was conducted. As a result, a generic ITS development architecture (GeNisa) has been proposed, which combines the features of knowledge base systems (KBS) with object-oriented methodology. The GeNisa architecture consists of the following components: a tutorial events communication module, which encapsulates the interactive processes and other independent computations between different components; a software design toolkit; and an autonomous knowledge acquisition from a probabilistic knowledge base. A graphical application development environment includes tools to support application development, and learning environments and which use a case scenario as a basis for instruction. The generic architecture is designed to support client-side execution in a Web browser environment, and further testing will show that it can disseminate applications over the World Wide Web. Such an architecture can be adapted to different teaching styles and domains, and reusing instructional materials automatically can reduce the effort of the courseware developer (hence cost and time) in authoring new materials.

GeNisa was implemented using Java scripts, and subsequently evaluated at various commercial and academic organisations. Parameters chosen for the evaluation include quality of courseware, relevancy of case scenarios, portability to other platforms, ease of use, content, user-friendliness, screen display, clarity, topic interest, and overall satisfaction with GeNisa. In general, the evaluation focused on the novel characteristics and performances of the GeNisa architecture in comparison with other ITS, and the results obtained are discussed and analysed.

On the basis of the experience gained during the literature research and GeNisa development and evaluation, a generic methodology for ITS development is proposed as well as the requirements for the further development of ITS tools. Finally, conclusions are drawn and areas for further research are identified.

ACKNOWLEDGEMENTS

This work has benefited from the support of many people. I would like to thank my supervisor and mentor Dr Vlatka Hlupic, for her insight and for encouraging and supporting me throughout this research. She has provided just the right mix of guidance, feedback, patience and motivation. and for this I am most grateful.

I am also grateful to Prof. Ray J. Paul for his valuable feedback.

I would also like to express my gratitude to different organisations and individuals that participated in the empirical studies.

I am grateful to my family for their support and patience throughout.

DECLARATION

Some of the material contained in this dissertation appears in the following publications:

Atolagbe, T. and Hlupic, V. (1998). A Conceptual Model for An Internet Based Intelligent Tutoring System for Simulation Modelling. *In the Proceedings of 10th European Simulation Symposium and Exhibition.* Bargiela, A. and Kerckhoffs, E. (Eds.), Nottingham, United Kingdom, pp. 692–694.

Atolagbe, T. (1997). A Generic Architecture for Intelligent Instruction for Simulation Modelling Software Packages, *Informatica* 21, pp. 643–650.

Atolagbe. A. T. and Hlupic, V. (1997). A Reusable Architecture for Intelligent Tutoring Systems for Teaching Simulation Modelling. *In the Proceedings of the 9th European Symposium on Simulation in Industry Conference.* Kaylan, A.R. and Lehmann, A. (Eds.), Passau, Germany, pp. 187–192.

Atolagbe, T. and Hlupic, V. (1997). SimTutor: A Multimedia Intelligent Tutoring Systems for Simulation Modelling. *In the Proceedings of the Winter Simulation Conference.* Andradottir, S., Healy, K. J.. Withers, D. H and Nelson, B. L. (Eds.), Atlanta, Georgia, pp. 504–509.

Atolagbe, T. and Hlupic, V. (1997). Interactive Strategies for Developing Intuitive knowledge as Basis for Simulation Modelling Education. *In the Proceedings of the Winter Simulation Conference.* Andradottir, S., Healy, K. J. Withers, D. H., and Nelson, B. L. (Eds.), Georgia, pp. 1394-1402.

Atolagbe, T. and Hlupic, V. (1997). Intelligent Multimedia Tutoring for Simulation Modelling Education. *In the Proceedings of European Simulation Symposium*, Istanbul, Turkey, Kaylan, A.R. and Lehmann, A. (Eds.), pp. 280-284.

Atolagbe, T. and Hlupic, V. (1996). A Generic Architecture for Intelligent Instruction for Simulation Modelling Software Packages, *In the Proceedings of Winter Simulation Conference*, Charnes, J. M, Morrice, D. J., Brunner, D. T., and Swain, J. J. (Eds.), pp. 856-863.

Atolagbe, T. and Hlupic, V. A Generic Architecture for Instructional Systems for Simulation Modelling. (1996). *In the Proceedings of European International Conference on Simulation Modelling,* Budapest, Javor, A., Lehmann, A. and Molnar, I. (Eds.), pp. 389-393.

TABLE OF CONTENTS

Title Page Page

ABSTRACT II
ACKNOWLEDGEMENTS III
DECLARATION IV
TABLE OF CONTENTS VI
LIST OF FIGURES XI
LIST OF TABLES XII
LIST OF ABBREVIATIONS XIII

CHAPTER 1. INTRODUCTION 1
1.1 INTRODUCTION 1
1.2 RESEARCH AREA 4
1.2.1 Intelligent Tutoring Systems 6
1.2.2 Addressing the Knowledge Acquisition Bottleneck 7
1.2.3 Component-Based Architecture 8
1.2.4 Pedagogical Agents 9
1.3 RESEARCH OBJECTIVES 10
1.4 RESEARCH MOTHODS 11
1.5 DISSERTATION OUTLINE 13
1.6 SUMMARY 14

CHAPTER 2. BACKGROUND RESEARCH MATERIAL 16
2.1 INTRODUCTION 16
2.2 INTERACTIVE INSTRUCTIONAL ENVIRONMENT 16
2.3 INTELLIGENT TUTORING SYSTEMS 18
2.3.1 Intelligent Tutoring Systems Components 19
2.3.2 Case-Based Intelligent Tutoring Systems 26
2.4 KNOWLEDGE REPRESENTATION METHODS 27
2.4.1 Knowledge-Based Hypermedia 29
2.4.2 Knowledge Acquisition 31
2.4.3 Probabilistic Reasoning 32

2.5 CRITIQUE OF THE LITERATURE 34
2.6 COMPARISON WITH OTHER RELATED RESEARCH 39
2.7 SUMMARY 40

CHAPTER 3. A GENERIC ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS 42
3.1 INTRODUCTION 42
3.2 THE ORIGIN OF THE GENERIC ARCHITECTURE 43
3.2.1 Designing Users' Interactivity 44
3.2.2 Knowledge Support Components 48
3.2.3 Requirements for a Generic Intelligent Architecture 48
3.2.4 Software Architecture 51
3.2.5 Consistency Constraints 53
3.3 METHODOLOGICAL ASSUMPTIONS 54
3.3.1 Knowledge Sharing and Reuse 56
3.3.2 Component Behaviour 57
3.4 KNOWLEDGE REPRESENTATION 58
3.4.1 Components and Interoperability 59
3.4.2 Knowledge Base 62
3.4.3 Instructional Planning 67
3.4.4 Discourse Planner 69
3.4.5 Teaching with Case Scenarios 70
3.5 INTELLIGENT TUTORING SYSTEM ARCHITECTURE 72
3.5.1 Presentation System 72
3.5.2 Domain Expert 73
3.5.3 Inference Engine 75
3.5.4 Pedagogy Model 76
3.5.5 Student Model 77
3.6 KNOWLEDGE ACQUISITION 77
3.6.1 Integrating Pedagogical Agents 78
3.6.2 A Web-Based Instructional Environment 80
3.7 SUMMARY 84

CHAPTER 4. DEVELOPMENT OF THE GENERIC INTELLIGENT INSTRUCTIONAL SYSTEM (GENISA) 85
4.1 INTRODUCTION 85

4.2 DESIGN ENVIRONMENT FOR THE GENERIC ARCHITECTURE 85
4.2.1 Unified Modelling Language 86
4.2.2 Design Environment Class Library 91
4.2.3 Components Events Manager 93
4.2.4 Design Components 95
4.3 GENISA LEARNING ENVIRONMENT 99
4.3.1 User Interface 101
4.3.2 Remedial Planner 101
4.3.3 Text-to-Speech Engine 103
4.3.4 Pedagogical Agents 103
4.3.5 Inference Engine 104
4.4 KNOWLEDGE ELICITATION AND TUTORIAL MANAGEMENT 104
4.4.1 Pedagogical Technique 105
4.4.2 Student Model Components 108
4.4.3 Tutorial Hints 110
4.4.4 Reference Library 111
4.4.5 Knowledge Acquisition Module 112
4.5 A TUTORIAL FOR SIMULATION MODELLING: AN EXAMPLE 113
4.5.1 Student Activities During Instruction 117
4.6 Deploying Applications over the World Wide Web 119
4.7 Summary 120

CHAPTER 5. EVALUATION OF THE GENERIC INTELLIGENT TUTORING SYSTEM ARCHITECTURE (GENISA) 122
5.1 INTRODUCTION 122
5.2 CONDUCTING EVALUATION OF THE GENERIC ARCHITECTURE 123
5.2.1 Evaluation Objectives 124
5.2.2 Evaluating the Generic Architecture 124
5.2.4 Evaluating the User Interface 126
5.2.5 Evaluating the Implementation 127
5.2.6 Assessing the Design of the Generic Architecture 127
5.2.7 Informal Evaluation 129
5.3 USER TRIAL 129
5.3.1 Questionnaire 129
5.3.2 Evaluation Criteria 130
5.3.3 Evaluators 130

5.3.4 Results and Interpretation of the Questionnaire 131
5.4 HEURISTIC EVALUATION 135
5.4.1 Procedure Used 136
5.4.2 Results of Heuristic Evaluation 136
5.4.3 Observations and Comments 137
5.4.4 Heuristic Evaluation: Concluding Remarks 138
5.5 CRITIQUE AND SYNTHESIS OF EVALUATION RESULTS 139
5.6 SUMMARY 141

CHAPTER 6. REFLECTION 143
6.1 INTRODUCTION 143
6.2 THE GENERIC ARCHITECTURE 143
6.2.1 Generic Architecture for Intelligent Tutoring System 144
6.2.2 Reusable Components for Intelligent Tutoring System 145
6.2.3 Knowledge Base 147
6.2.4 Automated Knowledge Acquisition Module 148
6.3 A METHODOLOGY FOR INTELLIGENT TUTORING SYSTEM DEVELOPMENT 149
6.3.1 The Concept 149
6.3.2 The Proposed Methodology for Developing Intelligent Tutoring Systems 151
6.3.3 Usability of the Proposed Methodology 160
6.3.4 Trade-offs Between Traditional and Proposed Methodology 162
6.3.5 Evaluation of the Proposed Methodology 165
6.4 IMPROVEMENT PROPOSALS FOR INTELLIGENT TUTORING SYSTEMS 166
6.4.1 Architecture Requirements 167
6.4.2 Implementation Requirements 169
6.4.3 Security and Constrained Execution 171
6.4.4 Dynamically Programmable Student Model 171
6.4.5 Reasoning with Multiple Diagnostic Components 172
6.5 SUMMARY 174

CHAPTER 7. SUMMARY AND CONCLUSIONS 175
7.1 INTRODUCTION 175
7.2 SUMMARY OF CHAPTER CONTENTS 175
7.3 CONCLUSIONS 177
7.3.1 Components Portability and Reusability 177
7.3.2 Knowledge Representation 179

7.3.3 The Generic Architecture 180
7.3.4 Facilitating Design Processes 181
7.3.5 Summary of Conclusions/Contributions 183
7.4 FURTHER RESEARCH 186
7.4.1 Instructional Factors 187
7.4.2 Development Environment 187

REFERENCES 189

APPENDICES 230
APPENDIX A. BACKGROUND RESEARCH MATERIALS 231
APPENDIX B. BAYESIAN NETWORK 247
APPENDIX C. EVALUATION CRITIERIA 260
APPENDIX D. EVALUATION QUESTIONNAIRE 267
APPENDIX E. EVALUATION DATA 277
APPENDIX F. HEURISTIC EVALUATION 280
APPENDIX G. INFORMAL ENQUIRIES 282

LIST OF FIGURES

Figure 1.1 Main Research Methods 12
Figure 2.1 ITS Components 20
Figure 3.1 Observer Pattern 53
Figure 3.2 Knowledge Representation 58
Figure 3.3 A Component Diagram 59
Figure 3.4 Main Modules of the Generic Architecture 62
Figure 3.5 Instructional Design Process 64
Figure 3. 6 GeNisa Architecture 73
Figure 3.7 Web-Based Learning Environment Components 83
Figure 4.1 Model of GeNisa Design Environment 87
Figure 4.2 Case Diagram for a Courseware 88
Figure 4.3. Structural Model Diagram of a Tutoring System 89
Figure 4.4 A Sequence Diagram for a Tutor 90
Figure 4.5 Section of GeNisa Design Environment Class Library 92
Figure 4.6 Events Communication Subsystem 94
Figure 4.7 Screenshot of GeNisa Development Environment 97
Figure 4.8 Design Assistant 98
Figure 4.9 GeNisa Learning Environment 101
Figure 4.10 Tutorial Remediation 102
Figure 4.11 Pedagogical Model 107
Figure 4.12 Structure of GeNisa Student Model 110
Figure 4.13 GeNisa Reference Library 112
Figure 4.14 Scenario Events Model/ Operations 115
Figure 4.15 Simulation Applets 116
Figure 4.16 Client Interview 118
Figure 4.17 Case Diagnostic Tools 118
Figure 4.18 Quiz Tool 119
Figure 4.19 WWW Interface 120
Figure 6.1 Schematic Representation of the Proposed Methodology 151
Figure 6.2 Proposed Methodology for ITS Development 155
Figure 6. 3 Abstraction Method for a Domain-Specific Application 156

LIST OF TABLES

Table 2.1 Examples of WWW-Based ITS Authoring Software 30
Table 4.1 A Summary of Advantages of the GeNisa Design Environment 99
Table 5.1 Description of Evaluation Groups 131
Table 5.2 Performance Score 132
Table 6. 1 Summary of the Trade-offs between Traditional and Proposed Methodology .. 164
Table 6. 2 Evaluation of the Proposed Methodology 165
Table 6. 3 Summary of Proposals for Improvement of ITS 173
Table 7.1 Research Objectives and Outcomes/Contributions 184

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CAL	Computer-Aided Learning
CAI	Computer-Aided Instruction
CLOS	Common Lisp Object System
COM	Components Object Module
CORBA	Common Object Request Broker Architecture
DAG	Directed Acyclic Graph
DCOM	Distributed Common Object Model
GFP	Generic Frame Protocol
GUI	Graphical User Interface
ITS	Intelligent Tutoring Systems
UML	Unified Modelling Language
OLE	Object Linking and Embedding
OMT	Object Modelling Technique
OMG	Object Management Group
OO-DBMS	Object-Oriented Database Management System
OODM	Object-Oriented Development Methodology

CHAPTER 1

INTRODUCTION

1.1 Introduction

This thesis investigates potential methods for developing a generic architecture for Intelligent Tutoring Systems (ITS). In particular, it investigates the methods and techniques used for analysis, synthesis, and refinement (i.e. development) of the ITS. This chapter introduces the areas of ITS and addresses the use of computers in education in general. Artificial Intelligence (AI) has been applied to education in many different disciplines for widely diverse purposes (Murray, 1999), (Murray, 1996), (Anderson, 1990), and such application has been declared to be "inevitable" (O'Shea and Self, 1983). Nonetheless, despite ever growing focus on the computer as a dominant medium in the field of educational technology (Säljö, 1996), (Koedinger et al., 1995), (Ely, et al. 1988), there are different opinions concerning the use of AI in the context of teaching tools and some authors have criticised such an approach as being of poor quality (O'Shea and Self, 1983). Before an ITS can be employed, it must have a knowledge base from which to teach. That knowledge must be represented in a form to be useful both for researcher/developer standpoint and from the point of view of presenting knowledge to student (Anderson et al., 1992), (Self, 1999).

ITS use AI techniques for representing knowledge and carrying out interaction with a student (Murray, 1996), (Clancey, 1987), (VanLehn, 1988), (Yazdani and Lawler, 1986). For a computerised instructional system to be considered "intelligent," it needs to know what, when, and how to teach the subject matter (Wenger, 1987), (Ohlsson, 1987). That is to say, the tutoring system must have an expertise in the domain being instructed as well as the capability to assess how learners are progressing in their knowledge/skill acquisition. Additionally, as ITS may be considered as complex systems that are costly to develop, it is opportunistic to investigate these concepts within the context of the software engineering concept of reusability. This research investigates ITS with an emphasis on exploring the possibility of developing a reusable architecture for ITS development. This poses the following research question: is it possible to develop an interactive, dynamic, reusable and portable component-based architecture for ITS?

ITS when compared to conventional classroom instruction can improve achievement levels of students, when tutored on a one-to-one basis (Shute and Psotka, 1996), (Bloom, 1984), (Anania, 1983). In this method the student and the tutor collaborate in the process of instruction (Goodyear, 1991). Unfortunately, one-to-one tutoring is expensive, and there are limited skilled human tutors to do all of the necessary tutoring (Murray, 1990). Computer-based training (CBT) and computer-aided instruction (CAI) systems were the first such systems developed as an attempt to introduce computers into the classroom (Wenger, 1987). In these systems, the instruction was not adaptable to the learner's needs or abilities (Ohlsson, 1991). Although both CBT and CAI may be somewhat effective in helping learners, they do not provide the same kind of adaptive learning that a student would receive from a human tutor (Brusilovsky, 1998), (Murray, 1996), (Bloom, 1984). However, for a computer-based instructional system to be adaptable, it must have the ability to reason with the domain and the learner (Wenger, 1987). Therefore, there is a considerable scope for research in this direction.

Knowledge acquisition is widely acknowledged as bottleneck in building ITS (Murray, 1997), (Hoffman, 1987), which is exacerbated by the lack of skilled intelligent components developers (Bell, 1998), (Durkin, 1994), (Hayes-Roth et al., 1983). This knowledge acquisition bottleneck has hampered the wide spread use of ITS in schools and in organisations (Anderson, 1987), (Ohlsson, 1987), (Murray, 1997). The bottlenecks may be due to the lack of structure in the organisation of the knowledge base, which may hinder the development, maintenance and reusability of these systems. Furthermore, it may be attributed to the choice of implementation platform and inconsistencies in ITS architectures that uses different reasoning techniques and representation formalisms.

Constructing an ITS by reusing previously developed components has always been a subject of considerable interest to ITS research and development (for example, (Sparks et al., 1999), (Ritter and Koedinger, 1997), (Roschelle and Kaput, 1995) (Mizoguchi, 1995)). Ohlsson (1987) postulates that an ITS should provide moment-by-moment adaptation of both content and form of instruction to the changing cognitive needs of the individual learner (VanLehn, 1996b), (Voss et al., 1995). In order to achieve these objectives, the system must be adaptive to the student instructional needs and provide useful explanations of student's misconceptions (VanLehn, 1996a), (Smith et al., 1993).

This research investigates how ITS can use AI techniques to support ITS development activities. The extant research focuses on exploring the potential of various learning strategies and knowledge representation formalisms for instructional purposes. Another theme is the design of

reusable ITS components (Murray, 1999), (Sparks et al., 1999), (Ritter and Blessing, 1998). The major concern here is to design and implement reusable ITS components capable of facilitating communication between the different components of the system and the user. The related issues concern what type of knowledge formalism is required to support adaptation and what internal mechanism is needed to capture the knowledge and to enable adaptive functionalities (Murray, 1996), (Brusilovsky et al., 1996), (Brusilovsky, 1998). The investigation of these issues is guided by contemporary learning theories, software engineering principles and supported by empirical data (Self, 1990a), (Self, 1992).

Review of the current literature shows that there are too few ITS-related publications to make informed design decisions about ITS authoring tools and evaluation (Clancey, 1993). This may be attributed to the fact that ITS are "difficult and expensive to build" (Murray, 1998). Therefore, this research investigates the feasibility of increasing productivity in ITS development by reusing and extending ITS components on different domains and across platforms.

Components of an ITS consist of the following: (i) tools that contains a component with expertise in teaching (Murray, 1999), (Anderson and Pelletier 1991), (Wenger, 1987); (ii) tools that contains an explicit modelling of expert knowledge and cognitive processes to support learners during problem solving (Anderson et al., 1995), (Gertner et al., 1998), (VanLehn, 1988); (iii) diagnosis of students' knowledge (correct, incorrect, and missing), (Ohlsson, 1987), (Wenger, 1987); and (iv) detection of student errors and to provide feedback specific to those errors (Murray, 1996), (Shute and Psotka, 1996), (Gugerty, 1997), (Wenger, 1987). Burns and Capps (1988) identify these areas as components of an ITS and refer to these as the expert model, the student diagnosis module and the curriculum and instruction module. Anderson (1988) similarly identified knowledge representation and tutoring methodologies as areas suitable for the application of intelligence. Self (1988; 1999) suggests that these approaches are inadequate and do not "map well". This may be attributed to the fact that ITS are monolithic and there is no clear-cut architecture used for implementation (Self, 1999), (Wenger, 1987), (Yazdani, 1987).

This thesis investigates how component-based development approach (Koedingeret al., 1999), (Roschelle et al., 1999), (Ritter and Koedinger, 1997), (Roschelle et al., 1998), (Suthers and Jones, 1997) can use AI techniques for developing ITS. In the context of this thesis, a component is defined as an architectural building block, which provides a unit of independent module and functionality, i.e. not bound to a particular program, language or implementation (Orfali et al., 1996). As such, components can be implemented as objects or as compositions of collaborating objects, and packaged as independent pieces of code. This approach has been adopted as the

framework for sharing *semantics* (the nature of the compositions and the role of each component) across ITS components and their applications (Koedinger et al., 1999), (Koedinger et al., 1997), (Macrelle and Desmoulins, 1998).

Current implementations of component-based approaches are inadequate for building component-based ITS in which the components must respond interactively to the user's needs (Ritter and Koedinger, 1997). Furthermore, ITS have not achieved reuse and component reuse may be costly (Sparks et al., 1999), (Murray, 1996). Therefore, it seemed feasible to try an implementation of the component-based approach that would be more suitable for ITS and to investigate the potential benefits that could be achieved. This approach may provide methods for addressing the knowledge acquisition bottleneck (Wenger, 1987) that characterises ITS development.

The aim of this thesis is to investigate the requirements of a ITS authoring tool development methodology.

The remainder of this chapter discusses this research area and states the problem investigated in this research, together with justification for doing this research. A brief review of development of ITS is provided. This is followed by the motivation for this research, research objectives and outline of this dissertation.

1.2 Research Area

A significant number of ITS architectures have been suggested (Paul et al., 1998), (Siemer and Angelides, 1998), (Siemer et al., 1998), (Ritter and Blessing, 1998), (Atolagbe and Hlupic, 1997a; 1996), (Ritter and Koedinger, 1996), (Murray, 1996), (Leitch et al., 1995), (Doukidis and Angelides, 1994), (Major and Reichgelt, 1991), (VanMarcke, 1992), (Wenger 1987). These architectures provide three ITS components: navigation, content delivery and controls. The behaviour and interaction of these components usually depend on implementation methodology; and these architectures focus on one or more ITS component to varying details. Furthermore, some earlier systems tried to keep domain knowledge and teaching strategy independent in order to maximise reuse of the tutorial system in other domains, but they found this unsatisfactory for teaching purposes (Murray and Woolf, 1992), (Major, 1993). For example SCHOLAR (Carbonell, 1970) and WHY (Collins et al., 1975) emphasise knowledge representation (domain expert) and tutorial dialogues (pedagogy model). BUGGY (Brown and Burton, 1978), DEBUGGY (Burton, 1982), and PROUST (Johnson and Soloway, 1984) emphasise student modelling (student model).

MENOTUTOR (Woolf and McDonald, 1985) emphasises tutorial discourse strategies (pedagogy model and inference engine). These systems have static and shallow knowledge representation formalisms (Murray, 1996), (Atolagbe and Hlupic, 1996).

Research and development in ITS are generally based on a standardised method for application development and implementation (Self, 1999), (Murray, 1999), (Murray, 1997), (Wenger, 1988). This may be attributed to the difficulty of knowledge representation and reasoning (Musen, 1993), (Hoffman, 1987). Furthermore, most extant research may be hindered by the following factors:

i. The expert model is an implementation dependent model of expert problem solving knowledge, with limited interaction (Wielinga and Breuker, 1990), (Lesgold et al., 1989), (Anderson, 1988). These systems cannot easily accommodate new knowledge (Murray, 1997), (Hoffman, 1987). Therefore, the use of multiple methods of accomplishing a task, with varying degrees of skill, may help the learner's knowledge acquisition process (VanLehn, 1996a) and enables the learner to view learning as a continuing process (Vygotsky, 1978).
ii. The instructional contents may not be detailed and flexible enough to provide the support needed in real applications (VanLehn, 1996a), (Anderson et al. 1995). Furthermore, the instructional techniques may consist of "bugs", which may result in misconceptions of the application area (Brown and Burton, 1978), (Anderson, 1987), (Anderson, 1990).
iii. Instructional contents and problem-solving methods are the result of sequence of actions, which may help to achieve the instructional goals (Anderson, 1990), (Cohen and Feigenbaum, 1982). These approaches use algorithms to aid the elicitation of content knowledge, and their implementation are usually domain specific (Murray, 1996).
iv. In order to make ITS development more manageable, ITS are generally developed for specific domains (Wenger, 1987), (Ohlsson, 1987). Therefore the tools and domain contents are specific for that application (Murray, 1999), (Murray, 1997). Also, the individual components may have little access to the functionality of other components, and the components may use different representation methods and programming languages.
v. Maintenance of developed components is a great problem because the software functionality changes very rapidly as the domain changes (Orey et al., 1993). Therefore, when ITS components changes in such a way that it affects the conceptual objects, then the courseware must change as well.
vi. The student models are generally content specific (Lesgold et al., 1993), (Self, 1990). The student model should involve long-term attributes and should preserve the skills across a

range of tutorials. These attributes of the learners such as the student model, the pedagogic knowledge and performance profile should be preserved across a range of tutorials.

vii. The formalisms used to represent knowledge in some ITS lack organisation, which may hinder knowledge management and portability.

These observations demonstrate that there might be some benefits to examine and investigate the features that make ITS components difficult to reuse. Therefore, this research investigates whether it is feasible to develop an architecture for ITS that consists of reusable components for application development and for instruction. The aim is to develop a framework that will facilitate developer/learner activities, and to enhance the effectiveness of their activities (Murray, 1999), (Bloom, 1956). Furthermore, this research investigate ways by which different types of knowledge can be represented and in different ways in order to achieve portability and reusability (e.g. (Murray, 1998), (Wielinga and Breuker, 1990)).

This research investigates methods for addressing these limitations based on the premise that AI and education research should be capable of modelling and supporting learning processes and that these processes are different across domains.

1.2.1 Intelligent Tutoring Systems

The classic ITS model involves four distinct components: an expert model to provide domain-specific knowledge, a tutor model to provide pedagogical knowledge, a student model to provide some estimate of the student's knowledge state, and an appropriate human computer interface for both the domain and instructional components of the system (Clancey, 1987), (VanLehn, 1988), (Halff, 1988), (Burns and Capps, 1988), (Wenger, 1987), (Woolf, 1992).

Different ITS architectures have been described in the literature (Murray, 1999), (Self, 1999), (Shute, 1995). This research investigates the feasibility of using a component-based approach alongside conventional ITS methods in order to develop a generic intelligent architecture for ITS. This research also aims to investigate the viability of using artificial intelligence techniques (knowledge-based system, planning and heuristic rules), object-oriented software engineering, and component-based approach to produce an intelligent environment for developing ITS and for delivering instruction.

Part of the knowledge acquisition problem stems from the monolithic nature of ITS and the inability to integrate with other environments (Murray, 1997), (Shute, 1993). Therefore, ITS may

be conceptualised as consisting of several interdependent components such as the user interface, student model, etc. (Wenger, 1987), (Self, 1987), (Self, 1999), (Ohlsson, 1987) (Suthers, 1992), (Winkels et al., 1990), (Clancey, 1987), (VanLehn, 1988).

Researchers in ITS and other domains have worked towards the development of shareable and reusable problem-solving methods and knowledge bases (Chandrasekaran et al., 1999), (Chandrasekaran, 1988), (Chandrasekaran, 1986), (Walther et al., 1992), (Eriksson et al., 1996), (Wielinga et al., 1993). The aim is to reduce development and maintenance costs, and to build flexible, component-based systems that can be adapted to different environments (Murray, 1996).

In order to investigate the viability of such systems, this research explores the use of object-oriented methods (Booch, 1994) for facilitating the development of ITS components, and to enable systems to inter-operate with commercial software and Internet resources (Brusilovsky and Cooper, 1999), (Brusilovsky et al., 1996), (Ritter and Koedinger, 1995). This approach might help reduce the cost and time for developing an ITS (Murray, 1996), (Murray, 1997) and to enable greater collaboration between users, allow components and instructional material to be shareable between diverse applications across the Internet.

Researchers have built large-scale environments or toolkits for constructing ITS components. For example, KREST (Steels, 1990), VITAL (Shadbolt et al., 1993) and PROTÉGÉ-II (Puerta et al., 1992) are all architectures for the development of components and component-based systems, yet none of these can use components built outside their own environment. Therefore, an ITS authoring environment needs to incorporate tools for managing evolving software components and versions management (Murray, 1999). It appears that it could be reasonable to assume that the development of the generic architecture for ITS is feasible.

1.2.2 Addressing the Knowledge Acquisition Bottleneck

The use of interactive hypermedia in ITS is emerging as a popular and innovative medium for ITS developers and researchers (Wong and Chan, 1997), (Angeleidis and Tong, 1995), (Brusilovsky et al., 1996). Interactive hypermedia provide a more flexible approach to learning that departs from current classroom instruction. Brusilovsky (1997; 1998) argued that the ability to adapt ITS to students' background and knowledge had rendered the hypermedia approaches to be more effective as an educational tool. World Wide Web (WWW) based training and ITS represent an excellent integration of the advanced technologies. WWW browsers overcome many of the shortcomings of traditional Computer Based Training (CBT) packages by affording

platform-independent supports. Adaptive navigation with an individualised user model and training materials can easily be updated (Weber and Specht, 1997), (Brusilovsky et al., 1996).

Software tools have been developed to address knowledge acquisition bottleneck (Boose, 1988), but this approach is generally based on the use of contrived tasks in order to identify conceptual dependencies between components data (Murray, 1997). Also, using an authoring tool to build an ITS involves both knowledge acquisition and design processes (Murray, 1997). Therefore, it seems feasible to attempt an implementation of an ITS architecture that supports the overall ITS design process, in order to investigate the potentials that could be achieved.

This research explores how AI and Object Oriented (OO) techniques can be used to reduce the effort required for developing an ITS. In doing so, a framework is constructed in which all the advantages of object oriented design such as extendibility of architecture and reusability beneficial effects may be tested. To build this framework the issue of explicit specification of component mechanism, functionality and why it may be beneficial must be addressed.

1.2.3 Component-Based Architecture

In the component-based architecture (Jacobson et al., 1997), (Spewak, 1992), (Shaw and Garlan, 1996), (Booch, 1998), (Szyperski, 1998) the underlying design goal is to accommodate reusability and sharebility, and allows more effective composition of independently developed components across platforms (Roschelle et al., 1999), (Booch, 1998), (Roschelle and Kaput, 1997), (Cox, 1996). Component-based architecture also provides a set of building blocks for constructing interoperable components, applications and systems of applications. This framework is open, object-oriented, and supports multi-platform. It also allows users to take advantage of multiple programming languages (Szyperski, 1998), (Booch, 1996), (Jacobson et al., 1997).

ITS development requires significant programming skills (Major, 1995), (Murray, 1998). Furthermore, ITS components development and their interaction may be hampered either by the lack of a precise specification of an ITS architecture and/or by having it encoded in terms of implementation details. Therefore, reuse of development methodology may be hampered by the inability to precisely define a developmental framework.

The possibility of reusing ITS components may prove more economical (Sparks et al., 1999) for researchers and developers, but it also presents some software architecture challenges in order to ensure flexibility, interoperability, and scalability of components. Therefore, this research will

investigate the feasibility of using a component-based architecture for developing a generic architecture for ITS.

A component may encapsulate multiple classes (Booch, 1994), and can therefore be viewed as being analogous to conventional modules. Also, a class can serve as the building block from which application is constructed (Booch, 1994), (Rumbaugh et al., 1991). However, these modules can be viewed as classes, as they support the object-oriented mechanisms of encapsulation and inheritance. Additionally, the components offer a great degree of flexibility because they are generic with respect to the components from which they inherit.

There are some authoring systems that support components sourced outside of their own environments, such as HyperCard (Apple, 1997), (Apple, 1993), and AuthorWare (Macromedia, 1997). However, these systems are principally designed and mostly used as closed proprietary environments.

This research will explore component-based approach in the light of how it can be used to address the knowledge acquisition bottleneck that is inherent in ITS (Murray, 1997), (Musen and Schreiber, 1995), (Hoffman, 1988) by (i) focusing on specific application classes and (ii) developing component based modules for these applications. The components in the software architectures are generic and interact with one other component through an interfaces i.e. an inter-application communication manager. As needed, the generic components may be parameterised to simplify customisation for particular applications.

In contrast to traditional ITS components (Wenger, 1987) development, the components are designed from the outset to be composed from existing software components that can be used in a variety of environments. By revealing the shared components of different systems at the various levels of generality, component-based architecture may promote the design and implementation of components and subsystems that are reusable, cost-effective and adaptable (Booch, 1994), (Jacobson et al., 1997).

1.2.4 Pedagogical Agents

Pedagogical agents are a contemporary approach for making computer-based learning more engaging and effective (Johnson et al., 1998). A pedagogical agent is a type of autonomous agent that can be used in a learning environment to support pedagogical, communicative, and tutorial tasks (Clancey, 1983), (Carbonell, 1970).

Pedagogical agents consists of animated interface agents (André, 1999), (Ball et al., 1997), (Hayes-Roth and Doyle, 1998), and graphical user interface to provide an effective environment for learning (Lester et al., 1999). Pedagogical agents use knowledge-based learning environments (Carbonell, 1970), (Sleeman and Brown, 1982), (Wenger, 1987), to support instruction. Pedagogical agents combine these two areas of AI, in order to provide a more effective instructional environment (Lester et al., 1999), (Elliott et al., 1999), (Ritter, 1997).

Pedagogical agents provide means for learners to learn and practice skills in a virtual world, and can interact with students through mixed-initiative, tutorial dialogue (Hume et al., 1996), (Fox, 1993), (Carbonell, 1970), (Burton and Brown, 1982). The agent can be used to demonstrate how to perform a task (Rickel and Johnson, 1999) and assist the learner during problem solving tasks.

This research will also explore the feasibility of using pedagogical agents to support instructional environment and to increase the bandwidth of communication between students and their learning environment (Rickel and Johnson, 1999). Other pedagogical approaches such as RAP (Firby, 1994) and Soar (Laird et al., 1987) are agents that can seamlessly integrate planning and execution, and can readily adapt to changes in their environments. However, the need to support instruction imposes additional requirements on the developer. Therefore, in order to support instructional interactions, a pedagogical agent requires a deeper understanding of its domain, the rationale for carrying out a task, and relationships between actions, than would be needed to perform the task (Clancey, 1983).

1.3 Research Objectives

The specific objectives of this research are:

i. To review critically ITS published literature, with a particular focus on ITS components, reusability issues and interactivities. Then, to examine critically the issues associated with current ITS architecture from a generic architecture perspective, therefore establishing the basis for this research.
ii. Propose a methodology for developing Intelligent Tutoring Systems.
iii. To develop a prototype of a multi-platform learning and authoring environment, reusable for different scenarios and applicable to other domains. The objective is also to explore the possibility of developing a portable toolkit that will facilitate the development of ITS.

Modular independence is necessary so that one module of the ITS component can be altered or replaced without affecting the other components.

iv. To investigate how ITS can use case-based reasoning to represent knowledge and use those representations to monitor and reason about user learning processing and to guide remedial learning in order to improve the instructional processes.

v. To evaluate critically and empirically the effectiveness of the authoring and learning environments implemented.

vi. To identify required future developments of ITS.

1.4 Research Methods

Self (1999) asserted that ITS research is usually part of a multidisciplinary endeavour, and adopts several methodological paradigms (Galliers 1992), (Dillenbourg et al., 1996). Theoretical constructs, application design, implementation and refinement constitute the core activities of the discipline (Stolterman 1995), (Self, 1990). However methodological and theoretical problems are inherent to epistemological investigation and ITS development in general. Therefore, this research employs different research perspectives (Galliers, 1992), (Galliers, 1991), (Galliers and Land, 1987), (Klein et al., 1991) in order to perform research towards addressing those difficulties that characterises ITS. This research objectives will be investigated by using action research (Patton, 1990). Action research involves not just appreciation of a research methodology, but also epistemological rationale that supports the study (Patton, 1990). Moreover, this involves the links between different research activities, and includes empirical studies and conceptualisation (Gibbs, 1995), (Patton, 1990), (Simonsen, 1994), (Klein et al., 1991). Figure 1 depicts the main research methods used for the overall study. Essentially, this includes:

i. *Empirical Studies*. The first part of this research involves the review of published literature that spans different areas of ITS including various aspects of cognitive and instructional theories, AI, and software engineering thereby establishing a compressive background theory for this research. The reviewed literature will be critically examined, which helps in attainment of deeper and more thorough understanding of this research area. This helps to identify difficulties in ITS development process, ITS components, the techniques for modelling users' dialogue, and knowledge representation formalisms, which translate into conceptualisation; and to perform research towards addressing these difficulties, by designing and implementing a prototype that addresses these limitations.

An inherent part of the latter effort is to critically evaluate the prototype, to assess whether it actually does address these limitations and to identify advantages and weakness of the methods used for the design and implementation of the final product of this research. This research employs both user trials (Monk et al., 1993) and usability heuristics (Nielsen, 1994) evaluation methods. The evaluation process will seek to critically appraise each component of the system and the functionality of the component methods. This should provide solid basis for further appraisal and re-conceptualisation of design and implementation processes.

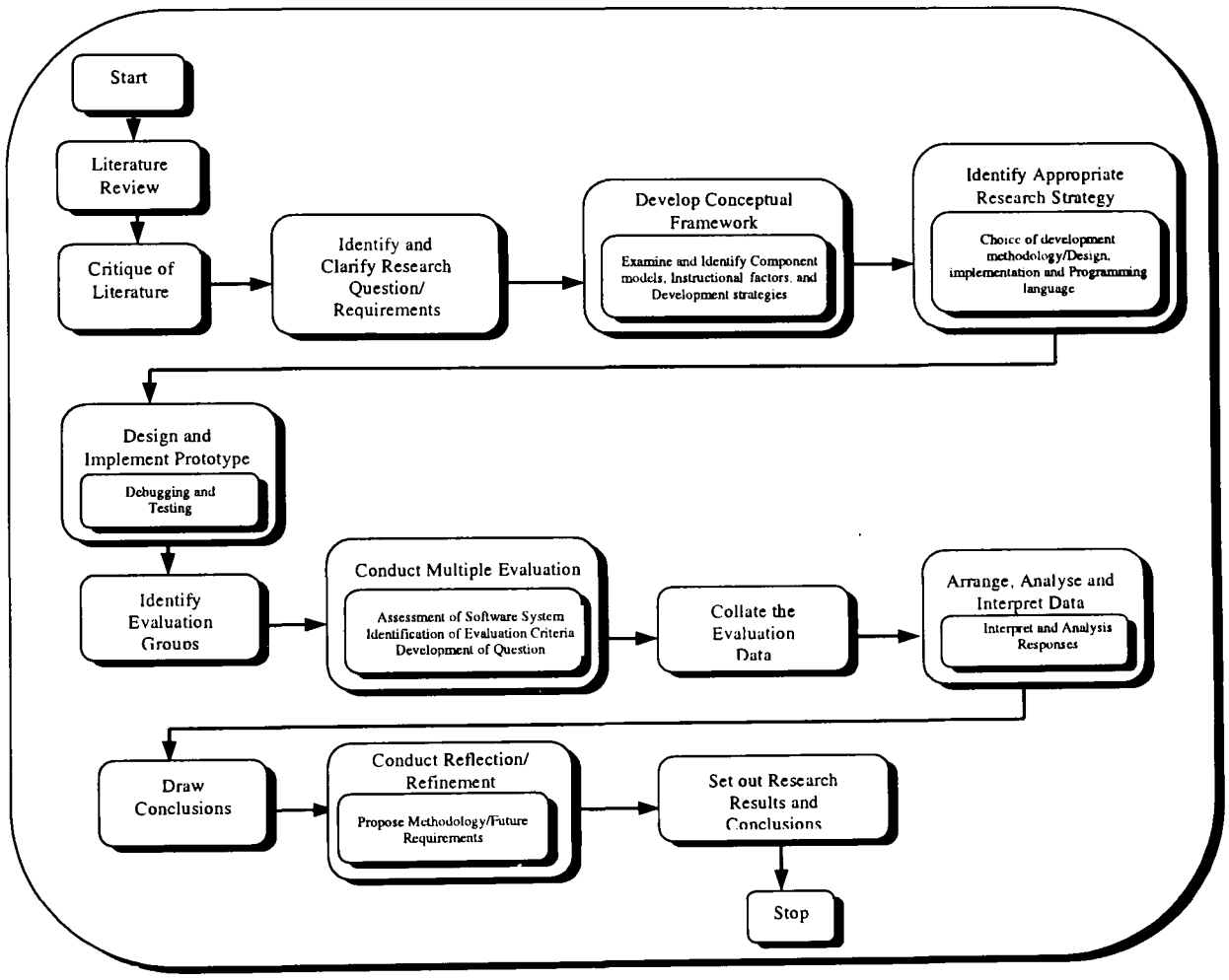


Figure 1.1 Main Research Methods

ii. *Conceptualisation.* The conceptual bases for this study are determined in view of epistemological rationales supporting existing theories (based on the reviewed literature), component-based approaches, knowledge representation formalisms and ITS development in the light of how to design and implement a prototype of a generic architecture for ITS. Essentially, this stage is responsible for establishing an informal, highly structured, specific mechanisms or notations and initial system specification (Th et al., 1993), (Rumbaugh et al., 1991) that is derived from reviewed literature. This approach facilitates the initial system development, to examine the interactions between components of the architecture

and to identify the required component functionality, without details of the design or implementation (Rumbaugh et al., 1991), (Garlan and Shaw, 1993). Furthermore, the resultant architectural structure and components will provide both framework and access to the necessary data and methods to permit development of consistent, interoperable, and reusable representations of the generic architecture.

The design and implementation of the generic architecture will take the form of guidelines and principles for software engineering (Sommerville, 1998), and object-oriented methods (Booch, 1994), (Rumbaugh et al., 1991) to support modularity, portability, and extendibility. This approach allows this research to examine critically how object-oriented methods and component-based development may be used to address the knowledge acquisition bottleneck inherent in ITS.

The conceptualisation, design and implementation of the generic architecture should be regarded as grounding in reality, for motivating and directing this research towards engineering reusable components for ITS, and for justifying the conclusions for this study more concretely.

1.5 Dissertation Outline

This chapter discusses the background, aims and rationale of this research.

Chapter 2 reviews the literature on Intelligent Tutoring Systems with reference to ITS development methodologies and component reuse, thereby identifying the problem domain for this research. Reviewed literature spans various fields of ITS, which helps to establish a comprehensive background theory for this study. Particular focus is placed on ITS components, instructional strategies, knowledge representation formalisms, and teaching with case-based reasoning. This chapter also presents critique of reviewed literature and comparison of the generic architecture with similar ITS.

Chapter 3 considers in more depth the benefit of a generic architecture for ITS. Based on literature review, this chapter presents the concepts of an open and reusable architecture and the underlying activities involved are analysed. The chapter argues for the desirability and feasibility of a domain-independent toolkit for ITS. The generality of the approach is illustrated by a consideration of its possible application in a variety of subject domains. On the basis of the literature, this chapter describes the conceptualisation of the generic architecture and its potential as a reusable, portable architecture for ITS.

Chapter 4 describes the generic architecture development and implementation. A functional description of each core component that populates the architecture is presented, along with its implementation methodology and interactions. This chapter also discusses other functionalies of the GeNisa environment within the contents of the architecture and development methodology. This chapter also demonstrates the practicality of the proposed architecture.

Chapter 5 describes the evaluation of GeNisa prototype. The usefulness of the development/learning environments, and the techniques used for the design and implementation were evaluated. The evaluation uses empirical and heuristic evaluation methods. The empirical method uses a questionnaire that was designed and distributed to different users, and feedback was obtained and analysed. The heuristic evaluation was based on guidelines obtained from literature to evaluate the elements of the system. This chapter also examines critically the data collected during evaluation and outlines the results of the study, in terms of the effectiveness of the architecture and whether GeNisa meets the requirements delineated in Chapter 3, together with the potential of the architecture to be extended and reused.

Chapter 6 reflects on the design and implementation of the generic architecture. This chapter extends and elaborates on the concepts of explicit planning and knowledge representation in the light of the results of the study. The design choices and trade-offs available for implementing the proposed methodology and for intermediate representations are considered. This chapter also proposes a new methodology for ITS development and identifies requirements for the further development of ITS. A comparison of the alternatives proposed was also provided.

Finally, Chapter 7 presents a summary of the work undertaken, and the conclusions that have been drawn. It also describes some of the potential areas for future research that have arisen from the research and development described in this thesis.

1.6 Summary

This chapter provides the main introduction to the issues covered by this research. Essentially, it has identified issues such as developments, limitations and current paradigms for ITS.

This chapter has also discussed how this research was conceived, the motivations and the research objectives. It has highlighted the areas of ITS, knowledge representation, pedagogical

agents and ways of addressing the knowledge acquisition bottleneck of ITS; and discusses some shortcomings of ITS. The chapter concludes by outlining the structure of this research.

CHAPTER 2

BACKGROUND RESEARCH MATERIAL

2.1 Introduction

The research presented in this dissertation draws on a number of different fields of study, which serves as a theoretical framework for this research and is subsequently conceptualised into software design and followed by implementation. The aims are to investigate how software engineering, cognitive psychology, AI, learning theories, and ITS field of study can be used to enhance the development of the learning and development environment; to critically appraise the characteristics, implementation and limitations of the reviewed systems; and to provide a critique of issues, methods and techniques raised from reviewed literature. The requirements for the generic architecture were derived from the reviewed publications. This was used to investigate the potential benefits of implementing an ITS by using the generic architecture.

Appendix A provides detailed background literature survey for this research. The rest of this chapter is organised as follows: Section 2.2 describes interactive instructional environments, Section 2.3 describes ITS development methodologies, followed by a review of knowledge representation methods in Section 2.4. This chapter concludes with a critique of literature and a comparison of the generic architecture for ITS with other relevant projects, and draws conclusions.

2.2 Interactive Instructional Environment

Interactivity is "a necessary and fundamental mechanism for knowledge acquisition and the development of both cognitive and physical skills" (Barker et al., 1998). Interactivity is important for the design of effective human computer interaction (Booth, 1989), (Norman and Draper, 1987), (Bunt, 1995). In the context of this research, interactivity is referred to as a method of providing mechanisms that allow users to browse, annotate, link and manipulate components (Ambron and Hooper, 1988). Schwier and Misanchuk (1993) introduced a detailed taxonomy of interactivity based on three dimensions: levels (reactive, proactive, mutual), functions (confirmation, pacing, navigation, inquiry, and elaboration) and transactions (keyboard, mouse, and voice). The "levels of interaction are based on the instructional quality of the interaction"

(Schwier and Misanchuk, 1993), which reinforces the idea that the higher the level, the better the instruction. The associated functions of interactivity include verification of learning, learner control, learner interrogation and performance support, and knowledge construction (Dickinson, 1995), (Schwier and Misanchuk, 1993).

Interactivity is essential during ITS development, as it lays emphasis on the ways in which users can access, manipulate and navigate through the instructional material (Brusilovsky 1996b), (Akpinar and Hartley, 1996). The interactivity adapted in this thesis is based on Akpinar and Hartley (1996) interactive methods. This provides a guide to different modes of communication between computer and learner. By applying these interactive "concepts" to ITS design, the various media elements can be integrated based on instructional decisions rather than visual appeal, allowing more effective communication and therefore potentially more instructional effectiveness (Jonassen, 1988), (Crawford, 1990). The following interactivity concepts were based on Akpinar and Hartley (1996) interactive classification. This is necessary in order to explore the potential benefits that could be realised for integration into ITS. It includes:

i. *Object Interactivity.* Refers to an application in which objects (buttons, icons, objects) are activated by using a mouse. When a user "clicks" on the object, there will be some form of audio-visual response. The functionality of such objects can be varied according to consequential factors, such as previous objects encountered, previous encounters with the current object or previous instructional performance/activity (Akpinar and Hartley, 1996), (Nelson, 1994).

ii. *Linear Interactivity.* Linear interactivity refers to applications in which the user is able to move (forwards or backwards) through a predetermined linear sequence of instructional material. This class of interaction does not provide response-specific feedback to learner actions, but simply provides access to the next (or previous) display in a sequence. This approach reduces the level of learner control (Schwier and Misanchuk, 1993), (Brusilovsky 1996b).

iii. *Support Interactivity.* One of the essential components of any software application is the facility for the user to receive performance support, which may range from simple help messages to complex tutorial systems. Support interactivity could be used to provide context-sensitive information during instruction (Brusilovsky 1996b), (Akpinar and Hartley, 1996).

iv. *Update Interactivity.* Relates to domain application components or events in which a dialogue is initiated between the learner and computer-generated content. Update

interactivity can range from the simple question-and-answer forms and learner feedback (Brusilovsky, 1996b).

v. *Construct Interactivity.* The construct class of interactivity is an extension to update interactivity, and requires the creation of an instructional environment in which the learner is required to manipulate component objects to achieve specific goals. Unless the construction was completed in the correct sequence, the task could not be completed. Construct interactions require significantly more design and strategic effort, as many parameters affect the successful completion of an operation (Akpinar and Hartley, 1996).

vi. *Simulation Interactivity.* Simulation interactivity extends the role of the learner to that of controller or operator, where individual selections determine the training sequence. The simulation and construct interactivity levels are closely linked, and may require the learner to complete a specific sequence of tasks before a suitable update can be generated (Akpinar and Hartley, 1996).

vii. *Non-Immersive Contextual Interactivity.* This concept combines and extends the various interactive levels into a complete virtual training environment in which the trainee is able to work in a meaningful, job-related context. Non-immersive contextual interactions require significant effort in design strategy and work well with a rapid prototyping methodology (Dickinson, 1995).

These interactive strategies provide a means for increasing the functionality of ITS and are useful for enhancing human computer interaction and for interactive learning (Brusilovsky, 1998), (Brusilovsky, 1996). Essentially, each interactive method should relate to the design of instruction and didactic strategy (Brusilovsky, 1998). The interactive strategies may allow learner to achieve the pedagogy objectives by performing one or more actions reiteratively (Bunt, 1995), (Schofield et al., 1994), (Podmore, 1991). These interactive elements form the basis for designing effective learning and development environments (Bunt, 1995), (Pea, 1993b).

2.3 Intelligent Tutoring Systems

Ohlsson (1987) has given a definition of ITS as a system that provides moment-by-moment adaptation of both content and form of instruction to the changing cognitive needs of the individual learner. ITS is used to address the deficiencies of computer aided instruction (Sleeman and Brown, 1982), (Hartley and Sleeman, 1973). In its role as a knowledge communication system (Wenger, 1987), ITS strives to optimise learning and problem-solving skills by means of adaptive, individualised instruction (Wenger, 1987), (Brusilovsky, 1998). Current approaches use

knowledge representation strategies and cognitive models of the student (Self, 1987), (Self, 1999), (VanLehn, 1988), (VanLehn, 1991), (Cristina et al., 1997). What distinguishes an ITS from a CAI system is the use of techniques such as explicit representations of knowledge, inferencing mechanisms and natural language dialogue (Wenger, 1987), (Clancey, 1987). One primary characteristic of ITS is using this knowledge for multiple purposes. For example, the same piece of knowledge might be used for different purposes (e.g. formulating a question and providing answer).

2.3.1 Intelligent Tutoring Systems Components

Three commonly used ITS components (Koedinger and Anderson, 1995), (Brusilovsky, 1995b) (Burns and Capps, 1988), (Wenger, 1987), consist of an expert knowledge i.e. *Expert model*, *Student model* which represents the student's current knowledge and experience and *Pedagogy model*, which represents an instructional knowledge (Wenger, 1987). The pedagogy model structures the interaction between the tutorial model and the student. The graphical user interface components control communication between the student and the system (Ritter and Blessing, 1998). ITS do not only describe knowledge, but they are also able to apply and evaluate learned knowledge (Wenger, 1987). Figure 2.1 illustrates the main components of an ITS. Various research activities have been directed towards different aspects of these components in order to focus on specific issues (Self, 1987), (Murray, 1996). The components of ITS are:

i. *Student Model.* The student model component is a representation of the student's knowledge (VanLehn, 1988), (Self, 1987), (Ohlsson, 1987). It describes the character and performance profile of the student's current state of knowledge and responses. Student models are occasionally used to refer to the data structure, and for recognising student plans or solution paths (Conati et al., 1997).

The student model provides the basis for diagnosing learner problems and selecting appropriate instructional mediation relating to tasks, problem-solving skills, methods and strategies (Siemer and Angelides, 1998), (Katz et al., 1993), (Lajoie and Lesgold, 1992), (Halff, 1988). ITS uses the student model in order to make instructional didactic decisions and diagnostic capabilities (Brusilovsky, 1998). The structure of the student model can be derived from (i) the problem-solving behaviour of the student, (ii) direct questions asked from the student, (iii) historical data (based on assumptions of the student's assessment of his skill level, novice to expert, Eliot and Woolf, 1995), (iv) the difficulty level of the content domain, and (v) constraints violated by the student (Ohlsson, 1992). The ITS compares the student's actual performance to the student model to determine if the student has mastered the content domain (VanLehn, 1988), (Self, 1987),

(Ohlsson, 1987), (Jameson, 1996), (Ohlsson, 1994). Advancement through the curriculum is dependent upon the ITS assessment of the proficiency level of the student (Self, 1987). The student model contains a database of student misconceptions and missing conceptions. This database is known as the "bug library" (VanLehn, 1988), (Brown and Burton, 1978). "A missing conception is an item of knowledge that the expert has but the student lacks. A misconception is an item of knowledge that the student has but the expert does not" (VanLehn, 1988).

Different approaches have been used for the student model. Semantic networks have been used to represent procedures and causal reasoning schemes; and for representation of the "Topic Network" (Murray, 1998). Procedures may be embedded into the network, which can use the semantic network to make inferences about the student model.

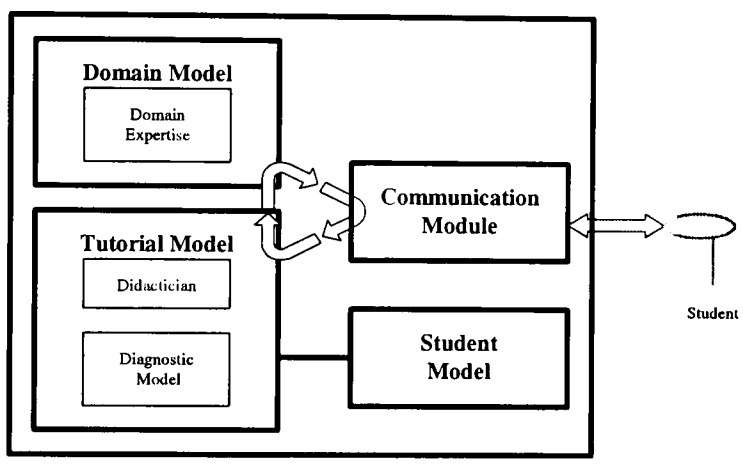


Figure 2.1 ITS Components

Student modelling has generally being modelled as an overlay model consisting of a subset of an expert model (Goldstein, 1982), (VanLehn 1988). The perturbation model was proposed (Brown and Burton, 1978) to account for students' misconceptions during instruction. These "perturbations" correspond to a set of expert rule representations in the domain. An alternative method for perturbation modelling uses artificial intelligence techniques to generate bugs or errors by systematically altering the rules that have been developed (VanLehn, 1991).

Currently, student models can be categorised into two varieties: overlay, where a student's knowledge is represented as a subset of an expert model, and model tracing where a student's concepts are represented as rules, and mistakes are attributed to buggy procedural rules (Anderson et al., 1995), (Anderson and Pelletier, 1991), (Anderson and Reiser, 1985), (Anderson, 1983). Model tracing technique may be used for comparing learners' performance with a pre-

existing domain expert model. This approach may encourage acquisition of new knowledge and practising the application of existing knowledge (VanLehn, 1996a).

Previous perturbation models have focused on perturbing procedural rules and attributing these to a student's mistakes (Langley et al., 1990), (Sleeman et al., 1990). A problem with this approach is that there is no deeper cognitive representation of a student's mistakes. Furthermore, fuzzy logic (Goodkovsky et al. 1994) and Bayesian networks (Collins et al., 1996) have been incorporated into the overlay model.

To date, cognitive learning theories have mainly focused on computational models of skill acquisition through the process of procedural compilation (Anderson, 1983), (Chan et al., 1992), (Mobus et al., 1993) and procedural chunking (Newell, 1990). In addition, procedural skill acquisition requires deeper cognitive structures for modelling abstract principles and concepts. Procedural skills are necessary for representing relationships among these abstract concepts through the use of causal links (Conati et al., 1997), (VanLehn, 1996a). In general, tutorial tasks may be acquired through repeated practice, which may improve performance.

Elio and Scharf (1990) and Eliot and Woolf (1995) developed a case-based student model that could dynamically represent a novice-to-expert transition based on a shift from focusing on surface features to abstract principles. This research work uses a similar case representation, using a network representation of the student's knowledge with nodes representing problem statements, surface features and abstract concepts, and annotated with appropriate case-scenarios.

Conati and VanLehn (1996) developed an extension to model-tracing (Anderson et al., 1995) diagnostic techniques that can recognise and reason about multiple problems solving styles in an ITS. These approaches are useful for understanding a student's overall problem-solving strategy and to provide a more interactive tutorial session. The model-tracing framework builds on the OLAE system (Martin and VanLehn, 1993), (Martin and VanLehn, 1995), which represents a student's conceptual understanding of a domain with a Bayesian network of concepts. Bayesian networks provide a probabilistic method for handling the uncertainties in student reasoning (Pearl, 1988a), (Conati and VanLehn, 1996).

Rosenblatt and Vera (1995) developed the GOMS (Goals, Methods, Operators, and Selection rules) framework for modelling mental states. The GOMS approach can be used to model either a system's internal knowledge for introspective reasoning, or a user's knowledge to do plan recognition and user modelling. Modelling the student's goals explicitly might add a useful

dimension to this research technique, by allowing this study to understand the student's knowledge acquisition processes. For example, a student who is learning new material will behave differently and require different types of feedback than a student who is reviewing previously learned material for a final examination.

Fox and Leake (1994) model case-based planning strategies as a set of expectations about system behaviour. They use failures in expectations to trigger diagnosis and repair of failures, by identifying new indices for case retrieval. Also, Eshelman et al. (1993) describe MOLE, a system of knowledge acquisition for heuristic problem solving. MOLE generates an initial knowledge base interactively, then detects and corrects problems by identifying "differentiating knowledge" that distinguishes among the SEEK system (Ginsberg et al., 1993) performs knowledge-based refinement by using a case base to generate plausible suggestions for rule refinement. Furthermore, case-based tutors (e.g. Schank et al., 1994), (Sherlock, 1991)) uses knowledge bases of case scenarios for tutoring. Case based tutors provide learners with "realistic scenarios" to support learning. This approach combines elements of traditional ITS paradigm with elements of simulation and case-based reasoning.

Mitchell et al. (1994) characterise a learning apprentice as an "interactive, knowledge-based consultant" that observes the student activities and analyses the problem-solving behaviour of users. One advantage of a learning apprentice is that it is running continuously as the system is used by different range of users. This research uses the method of learning by observing the user's choices during planning, which can be viewed as a learning apprentice. Thus, the evolving knowledge base reflects a broad range of expertise. The LEAP (Mitchell et al., 1992) apprentice uses explanation-based learning (EBL) techniques to explain and generalise cases (traces of the user's problem-solving behaviour) in the domain of digital circuits. A similar approach was used in DISCIPLE (Kodratoff and Tecuci, 1993), as well as similarity-based learning, to acquire problem-solving knowledge in the domain of design for the manufacturing of loudspeakers.

Non-determinacy is a significant problem in student modelling that has recently been addressed using probabilistic reasoning techniques (Hawkes et al., 1990), (Villano, 1992). Probabilistic reasoning has been used to introduce and reason about uncertainty in the student model by using a Bayesian approach (VanLehn, 1995). The design of student model depends on the domain knowledge representation (Murray, 1999), (Nawrocki, 1987), (Pirolli and Greeno, 1988) in order to provide appropriate student's diagnosis. It could infer that the student model component should provide adaptive remediation of instructional material as the cognitive needs of the learner change. This research uses Bayesian reasoning to introduce and reason about uncertainty in the

student model. A Bayesian approach used in this research is discussed in Chapter 3 and an implementation approach is described in Appendix B.

ii. *Pedagogy Model.* The pedagogy components depict the model of instruction and the tutorial contents, and the tutorial rules (Wenger, 1987). This model contains some simple or complex decision rules to determine the sequence and style of presenting instruction and feedback to the student. It also determines if the subject matter has been mastered or not. ITS actively interacts with student's inputs and diagnoses the student's level of understanding or misunderstanding of the knowledge domain. The tutorial exercises some control over the selection and sequencing of information, by responding to student questions concerning the subject domain and in determining when the student needs help and what kind of help is needed (Halff, 1988).

An effective ITS may meet the ever changing needs of the student. ITS diagnoses the student's characteristic weaknesses and adapts the instruction accordingly. As the student's level of proficiency increases, ITS will ideally conform to the evolving skill level of the student. ITS adapts as the novice evolves into a subject matter expert (Ohlsson, 1987), (Woolf and McDonald, 1985). Furthermore, ITS allow "mixed-initiative" tutorial interactions, where students can take the initiative in a tutorial dialogue (e.g. ask questions) and have more control over their learning (Wenger, 1987), (Cohen et al., 1998).

The pedagogy models can be designed by using ontology to represent tutorial actions and for specification of pedagogical events, and to represent tutorial tasks classification structure, (Mizoguchi et al., 1996), (Van Marcke, 1992), and (Murray, 1996). Pedagogy strategies may be embedded within an ITS, and the tutoring protocol determines the tutorial dialogue between the student and the pedagogy model.

The representation of an instructional strategy is also a vital attribute of an ITS. The instructional strategy is responsible for choosing an effective presentation method, providing student and instructional control, diagnosis of student's misconceptions, providing guiding remediation, user feedback and evaluating learners input and user problem solving activities and proving hints (Freedman and Rosenking, 1986), (Woolf and McDonald, 1984). The pedagogy knowledge has been represented by using production rules to represent problem solving behaviours (Goldstein, 1982). These rules monitor students' errors, deviation/correction, and refinement during instruction. The advantage of using production rules for pedagogical knowledge is that, it allows

rules to be easily modified as required. Pedagogy knowledge can be implemented by using a genetic graph (Goldstein, 1982). Each node on the graph represents a basic task that must be performed, before further refinement. The genetic graph helps ITS to make pedagogy decisions by facilitating the process of tutorial task collection and generation of multiple explanations (Goldstein, 1982). Also, semantic networks have been used to represent instructional tasks for pedagogy knowledge; for example, IRIS (Arruarte, et al., 1997), REDEEM (Major et al., 1997), EON (Murray, 1998). Essentially, these systems tend to have different representational schemes and uses different teaching strategies, which can be used for authoring domain specific knowledge.

iii. *Expert Model.* The "expert model" (domain model), contains the knowledge base that generates intelligent responses to the student queries during instruction. It consists of sets of rules and procedures that belong to the domain. The expert model has a global function that can guide the user towards an optimal response/solution to a problem (Wenger, 1987).

Anderson (1988) identifies three basic types of expert models. First, the "black box" model that represents the domain knowledge this is necessary for learner diagnosis. For example, the SOPHIE (Brown et. al., 1973) program provides the student with a simulation-based approach for troubleshooting electronic circuits using the SPICE simulator. The system evaluates the student's input and uses numerical processes to provide feedback. The second type of expert module is the "glass box" model, which may or may not represent human reasoning processes. Examples of these programs are the GUIDON (Clancey, 1979) and MYCIN (Shortliffe, 1976) system. The third type of expert module is the "cognitive model" type. The cognitive model simulates the human problem solving processes and reasoning (VanLehn et al., 1992), (Recker and Pirolli, 1995). In principle, cognitive models are all attempts to use ITS to teach skills and knowledge in ways that will facilitate the knowledge acquisition and transfer. The knowledge in these systems can take the form of procedural, declarative and qualitative (Anderson, 1982), (Anderson, 1983).

Brown (1990), Anderson, (1992), and Corbett and Anderson, (1990) suggest that current theories and architectures in ITS will have an impact upon learning theories, and vice-versa. The implications for the design of ITS are set out in a series of necessary features of the learning tools, which support the epistemological rationale. Brown's assertion that theories of learning and ITS design are related is supported by Anderson (1990), who believes that there is considerable research in cognitive psychology that can be used as a guide to the development of ITS. He states that the success of an ITS can depend on its ability to achieve "task decomposition", a process of

simplifying learning, and the monitoring of a student's belief by generating production rules (task analysis) for a restricted knowledge domain.

It can be inferred that instructional processes should be planned and controlled by the learner (Bramley, 1990). Instructions must be open and should be suitable for different levels of learners. It should cover both the theory and practice of teaching, and provide the opportunity for the learner to practise the skills that were taught (Anderson, 1987).

iv. *Communication Module*. The communication module provides an interface for the learner to interact with the other components of the ITS. This includes the presentation of instructional materials (represented as text, graphics, speech, sound, videos, etc) and for the student to interact with the various components.

Research from other disciplines such as the cognitive theories, usability guidelines and software-engineering principles are applicable (e.g. (Apple, 1993), (JavaLook and Feel, 1998)), thereby providing responsiveness, permissiveness, and consistency. This implies that user interface guidelines should be applied across all user interface tools (Apple, 1993) and should reduce short-term memory load where possible and augmenting their short-term memory with external memory provided by the tool (e.g. Shneiderman, 1998). The process of knowledge communication requires that the interface contain a *discourse model* to resolve ambiguities in the student responses (Wenger, 1987). Carefully designed user interface allows the learner to interactively control the instruction, and the learning content.

Model-based user interface generation tools has been used for ITS, e.g. HUMANOID (Szekely et al., 1993), MECANO (Puerta et al., 1994), and GENIUS (Janssen et al., 1993). Each of these systems is based on the notion of a generic *interface* that could be used across domains. These systems provides form/menu-based interface and they support a limited range of tasks. However, ITS researchers/developers need to reduces the risk of depending on incorrect and inappropriate user interface components by drawing from multiple cognitive theories and multiple user interface guidelines. Furthermore, most ITS are based on graphical or menu-based used user interfaces, but some ITS use natural language dialogues (e.g. (Carbonell, 1970), (Brown and Burton, 1975), (Brown et al., 1982)). For example SCHOLAR included rich natural language facilities that allowed it to understand most student questions and answers. The main problem with natural language interfaces is how to ensure the coherence of the dialogues during instruction; natural language representation is resource intensive (Anderson, 1986).

The design of the user interface for an ITS should assist with knowledge management and delivery of instructional materials, (Murray, 1999). Furthermore, it should provide components for developers to formalise and visualise their knowledge (Murray, 1999). Simplifying input through the use of templates, data entry forms, and pop-up menus is quite common. Furthermore, coherence of the user interface component functionalities can be achieved by uniformity of screen design, functionality, and conformity with the standard Macintosh or Windows for PC "Look and Feel". Differing modes of tutorial navigation should be supported.

2.3.2 Case-Based Intelligent Tutoring Systems

Case-based reasoning (CBR) has been used to support problem solving and learning (Holodner, 1993), (Schank, 1982). CBR is based on analogical reasoning (Gentner and Stevens, 1983), and from theories of concept formation, problem solving and experiential learning within philosophy and psychology (Schank and Leake, 1989), (Tulving, 1977). Case-based ITS contains multiple instructional strategies (Chu et al., 1995), which can help a novice learner to develop expertise by performing the tasks (Hutchins, 1995).

Case-based reasoning, such as the CYRUS system, (Kolodner, 1983) was based on dynamic memory model and MOP theory of problem solving and learning (Schank, 1982). It was basically a question-answering system. The case memory model developed for this system has later served as basis for several other case-based reasoning systems including PERSUADER (Sycara, 1988), CHEF (Hammond, 1989), JULIA (Hinrichs, 1992), CASEY (Koton, 1989).

The HYPO system (Ashley, 1990), and CABARET (Skalak and Rissland, 1992) combined case-based and rule-based system for reasoning in legal judgement. Koton (1989) studied the use of case-based reasoning to optimise performance in an existing knowledge-based system, where the domain (heart failure) was represented by a deep, causal model. This resulted in the CASEY system (Koton, 1989), in which case-based and deep model-based reasoning was combined.

Other attempts include a case-based learning apprentice system for medical diagnosis (Plaza and López de Mántaras 1990), and the use of case-based methods for strategy-level reasoning (Lopez and Plaza, 1993). The role of episodic knowledge in cognitive models has been investigated in the EVENTS project (Strube, 1990).

It seems reasonable to infer that ITS can use CBR to represent internal reasoning processes and to use those representations to monitor and to reason about user learning processing and to guide remedial learning in order to improve the reasoning processes (Schank, 1982).

2.4 Knowledge Representation Methods

ITS uses knowledge based system for representations of subject matter (Murray, 1996), (Anderson, 1988). The knowledge bases use different knowledge representation formalisms and have different inferential capacities (Murray 1996), (Clancey, 1990), (Clancey, 1991); and use different AI paradigms for knowledge representation (Reichgelt, 1991).

Knowledge-based systems enable ITS not only to solve problems like experts, but also to explain how they have solved the problem (Chandrasekaran, 1988), (Chandrasekaran, 1986). Efforts in this area have always been directed towards problem-solving methods and to enable knowledge to be reusable in a plug-and-play manner (Chandrasekaran et al., 1999), (Chandrasekaran, 1986), (Walther et al., 1992), (Eriksson et al., 1996), (Wielinga et al., 1993). The reuse of an existing knowledge base, even if it requires adaptation, ought to lead to significant savings in development.

Traditional ITS have poor knowledge representation methods (Murray, 1997), (Hoffman, 1987). However, there are many standard methods for knowledge representation and knowledge sharing (Lenat, 1995), (Gruber, 1993). Knowledge representation formalism should allow incorporating metacognitive skills, curriculum, and domain knowledge separately (Lesgold, 1988). This approach allows an ITS to tailor the tutorial knowledge to the aptitudes of the student (Lesgold, 1988) and facilitates the development of reasoning and cognitive capabilities.

Different systems are built with a variety of knowledge representation languages (e.g. CLIPS, LOOM, or Pascal, LISP), and require components to be in different formats. Also, researchers have built toolkits for constructing components that are interoperable (Steels, 1990). For example, KREST (Steels, 1990), VITAL (Shadbolt et al., 1993), and PROTÉGÉ-II (Puerta et al., 1992) are all architectures for developing components. But none of these can use components built outside of their own environment.

Knowledge engineering has been considered as technology for building expert systems (Chandrasekaran, 1986), (Walther et al., 1992). It has been used for eliciting expertise, organising

it into a computational structure, and building knowledge bases (Chandrasekaran et al., 1999), (Chandrasekaran, 1986), (Chandrasekaran, 1988), (Walther et al., 1992). Although rule base technology has dominated until recently (Mizoguchi et al., 1995), a new technology based on knowledge modelling has appeared, such as the Knowledge Analysis and Design Structure (KADS) project (Wielinga et al., 1992), PROTÉGÉ (Puerta et al., 1992), and MULTIS (Mizoguchi et al., 1995).

These systems are based on the idea of generic tasks (Chandrasekaran, 1986), (Chandrasekaran, 1988) and heuristic classification (Clancey, 1985). Current attempts are based on task ontology, which serves as a theory of vocabulary/concepts used as building blocks for knowledge-based systems (Perez and Benjamins, 1999), (Mizoguchi, 1993), (Mizoguchi et al., 1995), (Guarino and Giaretta, 1995). This research considers knowledge representation to consist of task ontology, which consists of the computational architecture of knowledge-based systems and domain ontology. Ontology representation provides a means for both analysing and synthesising knowledge-based systems for ITS (Mizoguchi, 1993). An ontology is an explicit "specification of a conceptualisation" (Gruber, 1994), (Gruber, 1993). An ontology for a domain describes the underlying structure for the domain task, which defines the semantic interpretation of the knowledge and type of tasks (Aben, 1993), (Wielinga et al., 1992), which can be used to develop an ITS.

The choice of knowledge representation formalism depends on both the type of knowledge and its instructional use. Knowledge representation methods should support cognitive diagnosis of the student tutorial activities and provide adaptive interpretation of their misconception. This implies that knowledge representation formalism should support the instructional strategies, and different levels of tutorial. Furthermore, an estimate of cognitive complexity for tutorial tasks should be maintained so that the user interface and tutorial remediation can reflect the student problem solving activities (Goldstein, 1978).

Object oriented methods have been used for knowledge representation, in order to allow knowledge to be used on heterogeneous domains. For example, Byte-sized Tutor (Bonar et al., 1986) is an object oriented ITS architecture in which domain knowledge classes, are organised into a hierarchical inheritance lattice. This approach allows the architecture to share its structure and inherit the functionalities of other components. Besides, it also allows multiple representation of various ITS components such as the user interface, inference engine and adaptive problem solving methods to be represented in the architecture. An important consideration for ITS researchers is to investigate flexibility of developing formalisms for representing the various

components in ITS. An object oriented method is the most natural way to represent domain knowledge and to define the complex data structure (Booch, 1994). Nevertheless, components of a system can be represented as a reusable collection of functions, which may enhance productivity, reliability and quality.

2.4.1 Knowledge-Based Hypermedia

Hypermedia systems provide a rich environment within which users can interact with a wide range of tutorial document types. Hypermedia are used in tutoring systems in many variations to add interactivity and context to instructional systems (Angelides, 1995), (Agius and Angelides, 1997), (Woolf and Hall, 1996). Expert systems and hypermedia are successfully used in different domains (Fontaine et al., 1994), (Silverman, 1992), (Gloor, 1991) and both can be considered as tools to present, influence and distribute knowledge. Examples of adaptive hypermedia systems in education include: ANATOM-TUTOR (Beaumont, 1994), CLIBBON (Clibbon, 1995), ISIS-TUTOR (Brusilovsky and Pesin, 1994), HYPERTUTOR (Perez et al., 1995), SYPROS (Gonschorel and Herzog, 1995), HYPADAPTER (Hohl et al., 1996), InterBook (Brusilovsky et al., 1996), and AST (Specht et al., 1997). On the basis of the literature, it can be inferred that much of the popularity of the hypermedia systems in educational domain may be attributed to its capability to convey large numbers of hypermedia information to learners in structured and cohesive ways.

Adaptive hypermedia textbook methodologies (such as (Brusilovsky, 1996 and 1998)) suggest building domain models and user models for adaptive HYPERBOOKS, by using semantic nets to describe these domain models and to index the HYPERBOOKS with the corresponding domain nodes.

A hypermedia document can be used as a helpful part of a tutoring system or it can be the tutoring system by itself. An example of a stand-alone hypermedia teaching system is the Hypermedia Tutor (Gloor, 1991), based on HyperCard. The system allows the students to explore the HyperCard stack with pictures, text, sound and videos. Furthermore, Pedro (The Spanish Tutor) (Angelides and Gibson, 1993) also uses HyperCard for ITS. A tool for building such special hypermedia-supported instructional systems is presented in Brusilovsky, (1998) and Murray et al. (1999).

TRAINER (Reinhardt and Schewe, 1995) is an ITS for diagnosis and the retrieval of symptoms of a patient. The presentation of symptoms and the links from the hypermedia information elements (pictures, text) to the knowledge base is done in the hypertext system HITS.

ELM-ART (Brusilovsky et al., 1997) and ELM-ART-II (Weber and Sprecht, 1997) systems teach LISP programming by providing an intelligent interactive integrated textbook. The system is hypermedia based and is adaptable to the student's problem-solving methods. ELM-ART dynamically generates all the HTML pages based on the student model. These approaches allow the systems to be able to deploy application over the Internet. Hypermedia systems are continuously being used to improve ITS for different domains with declarative, procedural knowledge bases (Brusilovsky, 1998).

Other WWW-ITS includes CALAT (Nakabayashi et al., 1997), AST (Specht et al., 1997), MANIC (Stern et al., 1997), Medtec (Eliot et al., 1997), and DCG (Vassileva, 1997). The systems provide discourse management of instructional materials over the WWW. Furthermore, WWW-based ITS allows component-based ITS development (Koedinger et al., 1999), (Roschelle et al., 1999), (Ritter and Koedinger, 1997), and (Roschelle and Kaput, 1996), and can allow developers to encapsulate, share and reuse ITS components across platforms. This approach can facilitate and encourage collaboration learning and social interaction among students in a virtual learning environment (Koenemann et al., 1999), (Hoppe, 1995), (Ikeda et al., 1997), (Dilenbourg et al., 1996).

Table 2 enumerates prominent WWW based ITS software and their domains. The list demonstrates how the supporting WWW technology, which links the ITS components together, provides the adaptive environment for instruction over the WWW. Furthermore, the structure of the Web allows dynamic updating of tutorial material and for dynamic invocation of components functionalities.

Table 2.1 Examples of WWW-Based ITS Authoring Software

SYSTEMS	DESCRIPTIONS
ELM-ART	Brusilovsky et al. (1996) supports learning programming in LISP and provides adaptive navigation with an individualised user model.
CALAT	Nakabayashi et al. (1997) courseware development package, supports curriculum sequencing of WWW.
AST	Specht et al. (1997) applies adaptive navigation support by sorting of student

	solutions on WWW.
InterBook	Brusilovsky and Schwarz (1997) uses adaptive navigational support to provide information to the user. Development environment for tutorial pages on WWW.
Medtec	Eliot et al. (1997) provides adaptive presentation by generate adaptive summary of book chapters.
DCG	Vassileva (1997) generates tutorial dynamically, according to the students' tutorial goals and previous knowledge. Also provides adaptive sequencing of tutorial over WWW.
WITS	Okazaki et al. (1997), (1996) an ITS for differential calculations
Belvedere	Suthers and Jones (1997) provides tutorial mapping environment by using node-link graphs representing logical, and relationships between assertions. Also uses the Java to support real interactivity.
Manic	Stern et al. (1997) provides hypertext component, and supports adaptive sequencing of instruction.

2.4.2 Knowledge Acquisition

There are growing interests in AI techniques in academia, industry and government (Barr and Feigenbaum, 1982). Therefore, terms such as "knowledge elicitation", "knowledge representation" and "machine reasoning" now have the same meaning (Barr and Feigenbaum, 1982).

Several researchers have developed task-specific architectures where specific problem-solving methods, such as propose-and-revise and cover-and-differentiate, are used to solve classes of problems, such as configuration and diagnosis (Chandrasekaran et al., 1999), (McDermott, 1988), (Chandrasekaran, 1988), (Chandrasekaran, 1986). Some model-based knowledge acquisition (KA) tools, such as PROTÉGÉ for the episodic skeletal-plan refinement (ESPR) method (Musen, 1989), and ROGET for heuristic classification (Bennett, 1985), use knowledge roles defined by these problem-solving methods as models for the domain knowledge to be acquired from the application expert. The propose-and-revise method, for example, defines parameters that have values, constraints on these parameter values, and fixes that modify parameter values when constraints are violated. These tools are based on monolithic problem-solving methods, and are difficult to extend when the domain tasks to be solved have requirements that do not fit exactly the capabilities of the methods. Moreover, the task requirements and their knowledge base requirements for these tools will change during the

lifecycle. Therefore, inflexible knowledge representation formalisms could make maintenance costly and difficult.

In order to engineer a flexible and robust toolkit, several research groups have developed architectures in which problem-solving methods are composed from small-grained reusable components (Birmingham and Tommelein, 1992), (Chandrasekaran et al., 1992), (Klinker et al., 1991), (Musen, 1989), (Steels, 1992), (Wielinga et al., 1992). Although these architectures share many similarities in their approach to composing problem-solving methods, they differ remarkably on the degree of support they provide to the task of acquiring domain knowledge from the application experts (Chandrasekaran et al., 1992), (Musen, 1992). The goal in the PROTÉGÉ-II project (Puerta, 1992) is to develop a framework for automating the generation of knowledge-acquisition tools, based on the knowledge roles defined by the problem-solving methods that are assembled and configured from reusable components (Musen, 1992). These efforts have resulted in the development of an environment that promotes reusability of problem-solving methods and their mechanisms.

To develop an application, the developer has to analyse the requirements of the problem to be solved, and select the method and mechanisms for the task and subtasks of the problem. The concept of developing an application from reusable ontologies is one of the primary goals for this research. This process involves incorporating the representation requirements of the various methods, and defines mapping relations that specify the correspondence between the knowledge in the ontology knowledge base (Perez and Benjamins, 1999), (Gennari et al., 1994), (Eriksson et al., 1994), (Eriksson et al., 1996) (Musen, 1992).

2.4.3 Probabilistic Reasoning

One of the fundamental gaps in the expressive power of standard knowledge representation paradigms is their inability to represent and reason with uncertain information (VanLehn, 1996b). Uncertainty may be unavoidable in an instructional environment and student modelling, where learner objectives and other information sources are invariably unreliable, and the learner's behaviour is unpredictable (Villano, 1992), (VanLehn, 1996b), (Sime and Leitch, 1993).

Reasoning with uncertainty is a common problem in knowledge representation to which many solutions have been proposed (Martin and VanLehn, 1995). Among these, the probabilistic framework is unique in its clear and coherent semantics, which supports fundamental operations

such as incorporating evidence from various sources and deciding on optimal courses of actions, including information-gathering actions (Duncan et al., 1994).

Traditionally, the barrier to the use of probability theory has been the complexity both of acquiring complex numerical knowledge and of reasoning with it. However, there is an emergence of a framework for representing probabilistic knowledge, by using Bayesian belief networks (Pearl, 1988b), (Pearl, 1993). Bayesian networks utilise the locality of the world, i.e. the fact that only a few attributes directly affect each other, to allow a concise and natural specification of complex probability distributions. The same representation also supports probabilistic inference algorithms (Pearl, 1988b). For example, probabilistic reasoning methods have been applied to student modelling (Conati and VanLehn, 1996), (Villano, 1992), (Martin and VanLehn, 1995), (Duncan et al., 1994), (Gitomer et al., 1995) to represent uncertainty in the student model.

Expert systems and uncertainty in AI are continuously influencing ITS development (Martin and VanLehn, 1995). Bayesian networks are one of the tools both for graphically representing the relationships among a set of variables and for dealing with uncertainties in expert systems (Pearl, 1988a), (Pearl, 1993), (Castillo et al., 1996). A key problem in Bayesian networks is attributes propagation (Pearl, 1988a), that is, obtaining the posterior distributions of variables when some evidence is observed. Several efficient methods for propagation of evidence in Bayesian networks have been proposed in recent years (Jensen et al., 1990). Each method exploits the independence structure contained in the network to efficiently propagate uncertainty (Kim and Pearl, 1983), (Lauritzen and Spiegelhalter, 1988), (Jensen et al., 1990), (Shachter et al., 1994).

However, both exact and approximate methods require that the joint probabilities of the nodes are specified numerically, that is, all the parameters must be assigned numeric values. In practice, exact numeric specification of these parameters may not be available, or it may happen that the subject matter specialists can specify only ranges of values for the parameters rather than their exact values. This research investigates the feasibility of using probabilistic representation in order to follow the student's reasoning dynamically. This approach can be used to generate hints and answer help requests during instruction.

2.5 Critique of the Literature

Each of the reviewed literature has different approaches to ITS development methodology and cuts across a range of domains with some commonality that could be extrapolated to as "best" or "desirable" features of an ITS, as well as establishing some important epistemological rationale for this study.

There is broader acceptance of ITS largely due to continuing interest in ITS research (Koedinger et al., 1997). This may be attributed to the use of contemporary technology such as WWW and Java applets to support ITS development and implementation. Furthermore, the traditional approach to learning and teaching can be counter-productive (Wenger, 1987). The main problem with developing an ITS is expense, and a new ITS may be difficult to integrate with existing courseware (Anderson, 1985), (Merrill, 1985). Authoring tools are required to leverage the development cost and time (Murray, 1999). Therefore, it seems reasonable to infer that there is a need for a shift in emphasis from a traditional development approach to a scaleable development method.

The main criticism about ITS is that they are "directive" in nature, because they are based on a diagnose-remediate model of tutoring (Self, 1992), (Self, 1987). ITS are also considered omniscient, because they attempt to detect all possible errors and misconceptions (Murray et al., 1990), (Graesser, 1993). Present research trends seem to indicate that researchers are moving away from building such applications and research is now focused on building reusable, intelligent learning environments that are adaptable to students needs (Dillenbourg and Self, 1992), (Murray, 1998).

There has been a lot of research into learners' behaviour and performance for example (Anderson, 1987), (Dillenbourg and Self, 1992). The majority of the reviewed literature describes the nature and characteristics of the domain, the empirical associations among variables that characterise the domain, the tasks and the curriculum. Some of these studies show that learning processes and cognitive development could provide a general framework for ITS development (Anderson, 1987), (Dillenbourg and Self, 1992), (Murray, 1998). This approach may provide several pedagogical benefits such as development of problem-solving skills (VanLehn, 1996), and to facilitate the development of instructional content with intrinsic interactive components.

ITS developers must put few barriers between learners and the tutorial system (Latham and Saari, 1979). Latham and Saari (1979) argue that ITS must increase responsiveness of the learner by

providing meaningful and context-sensitive help, video images of the tasks and learner control. Effective performance were be enhanced if the behavioural model of the learner were considered during the development stages. Latham and Saari (1979) suggest that behavioural role modelling may enhance performance.

The application of psychological theories in ITS development are, in addition to being an analytical technique (i.e. tasks classification, cognitive processes, learning styles, etc), it allows an adaptive instructional system to be designed, and learner behaviour to be represented (Latham and Saari, 1979), (VanLehn, 1988).

There are two problems with using only an overlay model for student modelling (Ohlsson, 1993), (VanLehn 1988). First, it assumes that the expert module is complete. However, it is possible for the student to employ a legitimate strategy that is not in the expert module. Second, overlay models do not address the situation where the student misuses or misunderstands information. It assumes that information is only present or missing (Wenger, 1987).

Although some of these reviewed research publications have been marginally successful, many ITS continue to operate solely in their own domains without the use of other applications after implementation (Koedinger et al., 1997). Furthermore, this survey of literature allows the explication of past and present ITS research perspective and to examine criteria for further studies.

Intelligent learning environments are currently the current research trend (Koedinger et al., 1997). These leaning environments, however, need to have all the components of an ITS to enable them to offer intelligent help to students and to carry out pedagogic activities. As a result, learning environments cannot avoid the complex ITS issues, such as student modelling and diagnosis (Ramadhan, 1992).

QUEST (White and Frederiksen, 1985), SODA (Soloway et al., 1991), DISCOVER (Ramadhan, 1992), (Ramadhan, 2000), and VIZ (Eisenstadt et al., 1992) are examples of such learning environments. These systems attempt to embody human cognition, especially some cognitive models of problem solving, such as a model of software design in mind, an explicit model of a virtual computing machine or causal models of the domain devices that are used by humans. The goal of supporting such "deep models" of a domain and its processes is to help learners to acquire a better understanding of the domain, so that they learn faster and better.

There are ample examples from the literature showing the effects of fragmented ITS development, and the failure to integrate the development and deployment processes. Although a multidimensional approach is strongly present in these publications, the development methodology provides limited extensibility and portability (if (Musen, 1992), (Musen et al., 1995), (Gennari et al., 1994)). Therefore, a common development perspective should help facilitate ITS design and implementation across domains.

Most of the surveyed papers used pedagogy principles to guide the development and delivery of instruction (Anderson, 1987), (Wenger, 1987). This method may be ideal for classroom teaching, but not ideal for an individual learner (Ohlsson, 1993), because individuals may be exercising different cognitive processes, which requires both the content and style of the instruction to be adaptive. Both pedagogy and learner cognitive principles should be considered when designing instructional systems and should incorporate multiple tests. The incorporation of multiple tests into tutorial tasks could result in the development of transferable skills. Practice during instruction is important for development of skills and it allows the learner to pursue the tutorial in depth. By using both pedagogy and cognitive principles, different learning environments can be considered during the design process, and can enable the instructional system to be versatile and have a wide application area (Gagne et al., 1992).

From the preceding discussion it could be inferred that traditional learning principles applied to modern training or instructional settings would be effective and therefore of use during the development of the instructional system (Gagne et al., 1992). Similarly, the learner is not conceptualised as "passive" during training and the principles of reinforcement will "affect" learning (Bruner, 1966). The learner actively brings to bear old and new "schema" and strategies to understand a new task (Anderson, 1987). Consequently, instruction has to become more subtly engineered from a cognitive perspective. Secondly, it is difficult to focus only on behaviour or stimulus and reinforcement to explain performance. Learner feedback and knowledge of results remains one of the most effective variables to be manipulated during instruction. Some of the principles derived from reinforcement theory may be useful in instructional design. This research attempts to utilise theories of instruction during courseware development. Self-efficiency has caught the attention of the social psychologists, and there is literature on the use of this concept to link learning with performance (Bramley, 1990). If the learner's responses were followed by a satisfactory and immediate feedback, then the behaviour might be learned. Otherwise, if it were followed by negative response, it will not be reinforced (VanLehn, 1986b), (VanLehn, 1988), (Clancey, 1987).

Skinner's theory of reinforcement is the only form of learning theory that has had any real impact on the practice of training (Bramley, 1990). Other areas of Skinner's work include a schedule of reinforcement and setting behavioural objectives and setting the behaviour of the learner until it achieves the objectives. An adaptive learning environment should provide means for learners to repeat some tasks and generate appropriate responses that are pedagogically linked to current tasks. When situations resulting from a response increase the likelihood of a response being repeated, this consequence is called a *positive reinforcement* (Skinner, 1968).

The "learner control" involves presenting the learner with a structured view of the domain that is to be learnt (i.e. the target simulation software, satellite analysis, driver support system, etc.), and using appropriate tutoring material (Gagne, 1970). The learner interacts directly with the instructional environment in a structured manner. This approach of presenting a structured learning environment to the learner is suitable for teaching different subjects.

Most of the existing systems for commercially available application software tutorials do not use any instructional systems methodology such as student modelling, diagnosis or application model generators. The application/animated tutorial teaches some of the functionality of the application software and is not suitable for educational use, and the tutorial is not portable to other domains. This approach may impede the reusability of the tutoring system for different applications.

User manuals could be used as an alternative to teach the functionality of application software. User manuals for packages such as ProModel PC, Microsoft Windows, Lotus Smart Office use text and graphics to illustrate the components of the software. This provides an extremely valuable input to the teaching of software because the manual serves as reference (i.e. an application software functionality reference) and is easier to use. Some of these software packages use templates to provide a means of replicating common structures found within a hypermedia network. This allows information with a well-defined structure, such as reference manuals and training material, to be easily incorporated and linked with the system.

Finally, the learning or teaching of different tasks is done best using different methods (Gagne 1985). Bloom (1956) and Gagne (1985) suggested classifications of knowledge and learner behaviour, and assert that different types of knowledge require different types of learning or instructional methods. Therefore, task classification may be used to produce pedagogic classification of the tutorial, in which commands are laid out in a sequence that is logical for the learner.

The critique of the reviewed literature can be summarised as follows:

i. There has been much educational and organisational interest in finding computer-assisted methods of optimising student learning, skill development and interactive learning systems.

ii. Most of the publications give a considerable impetus to identifying ways of ensuring that the learners fully utilise the learning aids provided by the instructional system and achieve a positive transfer of skill. The principal emphasis is on providing the learner with skills and a working knowledge of the basic concepts involved.

iii. Some of the reviewed publications provide development environment/courseware authoring tools (e.g. (Murray, 1999), (Murray, 1996), (Elsom-Cook, 1991), (Elsom-Cook, 1990). These development environments are based on using a prerequisite domain network structure. The network contains static links to the tutoring strategies. This approach may constrain developers into using the same set of network architectures, which may be suitable for teaching factual knowledge in related domains. Furthermore, the domain network structure may provide a restricted adaptation of pedagogy dialogues during instruction.

iv. Cognitive modelling may enhance the ability of an ITS to customise the tutorial dialogue thereby providing a better leaner control during instruction. To gain a better understanding of student activities, it may be necessary to examine the *learning content* as part of engineering holistic and adaptive ITS.

v. Some of the research publications describe ITS that are not portable to other platforms.

vi. There is no critical evaluation of software with the users, and some of the systems may not be commercially viable.

vii. There is a general mismatch between learner's cognitive skills and tutorial delivery, thereby making learner control difficult.

viii. Most of the research publications do not provide a cross platform choice of subject and means of delivery. The subjects are directed towards acquiring skills and knowledge in a specific domain.

ix. None of the studies have a common, stable identifiable set of characters, which are portable across different domains.

x. There is a definite architecture or standard benchmark for ITS. Each reviewed publication used different models and implementation approaches.

xi. The student misconception during instruction may be corrected through remedial tutoring, where the system employs a number of teaching strategies appropriate to the nature of the content and the student's current knowledge.

All these issues are addressed in the development of the generic architecture presented in Chapters 3 and 4.

2.6 Comparison with Other Related Research

This section discusses related research publications within a context of ITS, and related disciplines. This contrast is necessary in order to have a coherent understanding of the many extensive research efforts on ITS. This section examines a few of these sources to compare and contrast this research with other efforts that precede this study.

i. EON. EON (Murray, 1998) is an ITS authoring environment, which allows developers to develop the tutoring system's interface from scratch, and to represent their instructional strategies. EON uses a knowledge-based paradigm to support multiple tutoring strategies and "meta-strategies". Eon uses semantic network to represent the "Topic Network", and for creating presentation screens, although, no guidance is given to help the author create effective tutoring strategies.

The EON system consists of an authoring environment that implements the standard ITS model. The systems architecture is defined in a modular fashion, specifying which modules are required for different applications (Murray, 1998). The Eon architecture is designed to allow developers with limited programming experience easily to encode knowledge about misconceptions, course design, exercises, and interfaces into each component of the Eon architecture.

ii. *ACQUIRE-ITS.* ACQUIRE-ITS (Schaefer et al., 1992) is an integrated environment for developing knowledge based systems combined with an ITS authoring tool. The system allows the developer to build knowledge base using the ACQUIRE tool, and to create an ITS based upon content of the knowledge base. The ITS uses a case-based approach that is based on the rules of the knowledge base to create cases. The system uses a rule-based system to generate queries about the student's mistakes, and the tutorial presentation is text-based.

This research is loosely related to this study, by reusing "semi-automatically" the contents of a the knowledge base as a domain expert for an ITS. However, in contrast to this research, ACQUIRE-ITS depends upon the structure and implementation of the knowledge base, whereas in this research both the contents of the knowledge bases and the ITS components are reusable.

Furthermore, ACQUIRE-ITS teaches procedural knowledge by using declarative methods. This approach may not be effective for teaching procedural knowledge because procedural skills can only be learned by practice (Anderson, 1987).

iii. *ITSIE* (Intelligent Training Systems in Industrial Environments). This research investigates the use of qualitative modelling techniques and multiple models of instruction (Sime and Leitch, 1993) for industrial applications. The ITSIE study has a set of tools for developing ITS (Sime and Leitch, 1993) and a specification methodology for a generic architecture. The development approach supports multiple methods of knowledge representation and for representing instructional strategies. ITSIE model of users' behaviour and knowledge is based upon Rasmussen's (1986) information processing theory. Some aspects of this approach have been adapted in this study during systems development and is described in a previous section.

ITSIE research differs from this research in that it is mainly an authoring system. One advantage of ITSIE approach is that it does not restricts the developer to any particular knowledge representation formalism. However, it restricts the developer to one of the implemented strategies. Also, the system requires the developer to encode knowledge (including the simulation, domain knowledge and pedagogical knowledge) in the ITSIE format (Sime and Leitch, 1993). The approach adopted in this research, in contrast, allows knowledge to be represented in a formalism that can be reused across different domains, without the need for any modification. This approach allows an upward transition of emerging knowledge about effective pedagogical strategies for automated instruction.

This research has some similarities with the EON (Murray, 1998) research goals, i.e. the need to reuse different component modules across platforms and for different domains. However, it differs from this research in that Eon is primarily an authoring environment. The developer must completely specify all exercises that are presented to the student, as well as define a complete misconception library that is used for diagnosing the cause of student errors.

2.7 Summary

This chapter discussed the literature review of the current ITS research and history, development, different types of ITS applications/components. It includes the identification of a list of instructional factors, knowledge representation formalisms and limitations, thereby providing a

comprehensive background theory for this research. This reflects the multidisciplinary skill that ITS spans. Some of the issues highlighted in this literature are identified for further investigation during this research. This includes strategies for development and knowledge representation, portability, reusability and selection of appropriate instructional method and hints. Various strategies for knowledge representation, interactivity and providing instruction have been discussed together with uncertainty representation. Areas for further investigation are identified and further discussed in subsequent chapters.

This chapter has provided an articulation of the problems in ITS, and has listed a number of notable ITS software applications that have been developed and their different architectures. This reflects the propensity of researchers towards specific ITS components.

Finally, this chapter concludes with a comparison of this research with other published literature. It can be inferred from the literature that the development of a generic architecture for ITS is feasible and there are potential benefits that might be realised for reusing ITS components with cross platform support. Furthermore, ITS development consists of integrated metaparadigms that has a wide range of dynamic activities. Each of these perspectives are essential in order for ITS to support instruction adequately and adaptively.

CHAPTER 3

A GENERIC ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS

3.1 Introduction

In view of the literature presented in the previous chapter, the initial part of the research has concentrated on two major issues. First, an in-depth literature review on topics of direct relevance to ITS component reuse, portability and reusability. Second, gaining familiarity with, and a working knowledge of, the research that had previously been undertaken by various research organisations and institutions. The literature review has focused on the architectural, design and implementation approaches, including an evaluation of authoring tools and an investigation of courseware metaphors. It illustrates the nature of the multidisciplinary activities involved and some technical issues facing the development of ITS.

A critical issue addressed in this chapter is the composition of components of the generic architecture. This is addressed in order to define the structure of the system as a whole and the interaction between components based on user actions. As stated earlier, this research considered ITS as consisting of several interdependent components. Developing these ITS components primarily involves three main tasks: the construction of the ITS framework, the engineering of the domain-specific knowledge and the development of mechanism for controlling the communication processes between applications. The benefit of developing the generic architecture for an ITS is to ensure that the application area is not restricted to any single domain, and that the ITS components are reusable and are capable of operating in a heterogeneous environment (Cox, 1996), (Tu et al., 1995).

Based on the literature review, this chapter investigates how to design the ITS architecture (Wenger, 1987) into an integrated, adaptable, and tailorable environment for developing an ITS, with domain-independent components. It aims to investigate whether these systems can be conceptualised into reusable component development and learning environments, and describes the design of the generic architecture that addresses the problems highlighted in the previous chapter.

This chapter starts by discussing the origin of the generic architecture and proceeds to discuss issues to be considered when designing interactive ITS. This is followed by a description of the components of the generic architecture.

3.2 The Origin of the Generic Architecture

A comprehensive background literature search was first conducted, which involves synthesising the reviewed literature and identifying the problem domain. During analysis of the literature, the criteria and characteristics of the developed applications were identified and further analysed. This synthesis formed the basis on which the generic architecture was conceptualised and the basis for implementation discussed in Chapter 4.

The literature review discussed in Chapter 2 is indicative of the need to develop portable and reusable ITS components (Murray, 1999), (Murray, 1998), (Sparks et al., 1999) (Ritter and Blessing, 1998). A components-based approach allows ITS components to be reusable and portable and independently upgrades components as required (Musen, 1998), (Musen et al., 1995), (Cox, 1996), (Szyperski, 1998). This approach allows this research to represent each component module of the generic architecture as a unit of independent functionality and to inter-operate within the architecture (Szyperski, 1998). Components-based approach allows the generic architecture to addresses the lack of compatibility of ITS components, allows cross-platform utilisation of the components and meets different domains/platforms requirement. The design and implementation of the generic architecture concept hinges upon the notion that a component specification can be supported by multiple implementations (Sparks et al., 1999), (Murray, 1999) and can be used across different platforms. Therefore, earlier ITS architecture developed during this research (Atolagbe and Hlupic, 1996, 1997a) was re-conceptualised. A generic intelligent tutoring systems architecture (GeNisa) was finally conceptualised as a heterogeneous collection of modular subsystems with shared components, and carefully defined interfaces.

The generic architecture has been influenced by broader interdisciplinary research literature (i.e. software engineering, cognitive theories and AI principles) which helped engineer an effective system (Pea, 1993a), (Perkins, 1993). Furthermore, the Internet as the emerging technology influenced the way generic architecture was conceptualised and developed (Brusilovsky et al., 1996), (Brusilovsky et al., 1997). This resulted in conceptualisation of an architecture, which uses client/server architecture for presenting applications over the WWW. However, the Internet based architecture is necessary for large-scale deployment of applications across different platforms; it

also provides the capability to use commercial services as opposed to private networks to bring together diverse, geographically dispersed sites; use different local network topologies and technologies (e.g. Ethernet).

The design of the generic architecture components relates to the Model-View-Controller pattern (Buschmann et al., 1996) and encapsulates multiple Java Class libraries (Gosling et al., 2000), (Arnold et al., 2000), which consist of different *modules*. Each module consists of multiple classes, which serve as "building blocks" for implementing different components independently and supports the object-oriented mechanisms of encapsulation (the ability to hide the internal structure and behaviour of the classes) and inheritance (Smaragdakis and Batory, 1998), (Booch, 1994), (Biggerstaff, 1994). However, the View and Controller are packaged into a single *role* (Gamma et al., 1995). A *role* embodies a separate aspect of the class's behaviour (VanHilst and Notkin, 1996) and is reused to decompose the components into different set of classes and a set of *collaborations*. A *collaboration* is a collection of sets of roles (VanHilst and Notkin, 1996), (Booch, 1994). This approach has several advantages: (i) it allows different interface packages to be layered on top of the abstract classes (classes that cannot be instantiated that is used to create other object)), (Booch, 1994), (Rumbaugh, et al., 1991); (ii) different components classes can inherit the behaviour and functionality of another class; and (iii) it allows the use of different API (Application Program Interface) to implement different functionalities of the components, and permits other applications to use the generic architecture API for different applications.

In this research, a considerable time has been devoted to designing and implementing the software components based on the generic architecture tenets. It was in the course of the development of this software that the following technical issues converged: component reusability, portability, components interface, compatibility and the evolving need to deploy ITS over the heterogeneous platforms. Some of these issues are discussed further in this chapter and in Chapters 4 and Chapter 6.

3.2.1 Designing Users' Interactivity

Designing or participating within learning environments requires using cognitive or perceptual skills (Clark, 1997), (Hutchins, 1995), (Kirsh, 1996). Therefore, designing interactivities is necessary in order to reduce the user's cognitive load, which may render the tasks easier, faster, or less error-prone. Furthermore, designing users' interactivities is important in order to understand the dynamic of interactivities between the users' tasks and the system (Hollan et al.,

2000). The objective is to explore the concept of interactivity, particularly as it applies to the design of ITS components.

Users' problem solving includes the use of mental models, together with other problem solving techniques (Barker et al., 1998), (Gentner and Stevens, 1983), (White and Frederiksen, 1985). A users' mental model relates to some models of the component and the world in context (Barker et al., 1998). For example, mental models of a physical object are the conceptual representations of the object that provide predictive and explanatory means to users in understanding the system and guide their interaction with the components (Sein and Bostrom, 1989), (Young, 1983). These approaches are sometimes referred to as normative domains (Gentner and Stevens, 1983), because devices or objects in these domains have physical representations. As a result, users can create an internal representation of the component. Furthermore, they can also associate structural and functional relations among the actual devices/objects and their components (Gentner and Stevens, 1983). Considering users' interactivities can offer a range of benefits by providing improved flexibility and consistency and allows this research to be able to use users' interactivities more reliably to assist in knowledge acquisition (Greeno et al., 1998). The rest of this section describes some relevant cognitive theories in software engineering perspectives, component development, and the implications of the theory on application development are outlined. Some of the theories are:

i. *Decision-Making*. Schoen (1993) suggests that developers make design decisions under uncertainty, and designs are conceived partially (Schoen, 1983). The design decision is usually made in the context of "events" (VanLehn, 1991). As the developer understands the situation evolves, their mental model of the problem situation improved, thereby improving the design (Guindon et al., 1987).

Supporting a designer's decision-making process can be beneficial during application development by using associative memory structure and access (Cofer, 1975), (Kurland and Pea, 1985). Developers can assist in making design decisions by providing tools that allows the user to visualise their design and to readily access the design structure quickly and readily (diSessa, 1993), (Mayer, 1981), (Sebrechts et al., 1990). For example, design aids such as checklists, templates, may be beneficial for making design decisions.

ii. *Structural Decomposition*. Structural decomposition is a common strategy for designing applications (Rumbaugh et al., 1991). However, in practice, developers perform tasks in a top-down, hierarchical order structure according to their cognitive loads (Guindon et al.,

1987), (Visser, 1990), (VanLehn, 1988), (Park et al., 1987) and understanding of the circumstances.

Therefore, the structural decomposition approach allows developers to design applications by minimising cognitive load (Park et al., 1987), although this approach has been challenged by the belief that the cognitive development process is cumulative and not necessarily hierarchical (Hoffman, 1997). The use of visual representation during design processes may help designers to represent all their activities visually, which can minimise cognitive load.

The use of cognitive features such as cueing, or prompts may provide support during application development (Beck, 1991). Cues also work as "perceptual organisers", and facilitate emphasising central concepts in a domain (Ausubel et al., 1978), (Beck, 1991). Prompting and reminding the developer during design processes may help them to discover gaps in their knowledge and hence find a suitable alternative.

iii. *Users Memory*. Applications overloaded with information often render the systems ineffective (Nelson, 1993). Therefore, the system should be designed so as to deliver information to the user according to their needs, goals, and expectations; and it should relate to the user's model (Allen, 1990). Application may be more effective if cognitive overload can be minimised in components that require multiple representations (Ellis and Hunt, 1993), (Anderson, 1996).

This theory implies that design components should provide cues that allow users to retrieve related memories (for example, help system, examples, and error message).

iv. *Information Processing*. The information processing capacity (working memory) of human beings is limited (Lewis, 1996), (Just and Carpenter, 1992), (Miller, 1965). Byrne and Bovair (1997) conducted an experiment that showed that increased working memory loads caused procedural errors in conducting complex user interface tasks.

One implication of this theory is that working memory is limited, and the processing of information should be sequential (Just and Carpenter, 1992), (Baddeley, 1994) and the tasks should be related. Working memory span should guide developers in making design decisions and in forming plans for component usage i.e. developers should reduce working memory load (Shneiderman, 1998), (Byrne and Bovair, 1997). This might involve provision of task relevant information and of visual cues.

iv. *Problem-Solving*. Kintsch and Greeno (1995) suggest that developers must bridge the gap between their mental model of the system and that of the user. The domain and problem-solving mechanism within that domain relate to the users' current problem solving needs (Pennington, 1987).

This implication of this theory is that it ensures that problem-solving activities are flexible. Therefore, design support tools such as such as data flow diagrams can assist the developer to identify elements and relationships in the current problem-solving activities. This approach may facilitate comprehension and helps problem solving; context representation should be task-dependent (Redmiles, 1993). Moreso, providing feedback in the context of a problem solving, may be beneficial to the developer because it allows the user to respond constructively.

v. *Visual Representation*. Visual representations of design components are more effective for communication. Design components that are represented visually can be quickly recognised (Petre, 1995), (Nelson, 1993).

The implication of this theory is that design components should take the user's perspectives into account and must follow the same notations.

vi. *User Interface Guidelines*. Usability guidelines are specific rules that define a given graphical user interface and heuristic for manipulation (Nielsen and Mack, 1994), (Nielsen, 1993), (Shneiderman, 1998).

Some of these guidelines provide the style guideline used in the development and evaluation of some of the components in GeNisa. Detailed discussions of some of the components are provided in Chapter 4. Therefore, based on analysis and synthesis, the following conclusions are drawn from the above theories:

i. In general, experienced users have better understanding of the problem domain than inexperienced users and their knowledge is better organised (diSessa, 1993). Structural models of the problem domains are more important than functional models for program comprehension (diSessa, 1993).

ii. Users, in general, use three classes of mental models, namely the structural models, the functional models and the distributed models, for program comprehension (diSessa, 1993). With clear understanding of the types of users' mental models the system is to

support, developers can design an effective, interactive and usable system (Newman and Lamming, 1995).

iii. Users' mental models are imprecise, and are continually changing during initial learning and during components development (Norman, 1993), (Sebrechts et al., 1990). The change of domain may affect the way user uses the systems, therefore, the system should be adaptable.

iv. A software architecture design and instructional environments should provide cognitive support for all the components.

v. Design features of all components must be consistent and easy to use (Nelson, 1993).

vi. Flexibility and visibility must be incorporated in the design and development of all components.

Some of these elements form the basis of the interactivity requirement for the generic architecture.

3.2.2 Knowledge Support Components

The GeNisa learning environment provides the learner with components, which they can use during instruction by reflecting on their own cognitive skills (Anderson et al., 1995). This approach can enhance instruction by involving the learner in the instructional processes and by helping students acquire meta-cognitive and domain skills. In this research, knowledge support components is defined as a framework consisting of "components", which consists of specific knowledge activities. Knowledge activities are autonomous actions, events and behaviour that are directly executed by the user. This approach allows the representation of content to be decoupled from the design of the presentation and navigational structure, in order to facilitate modularity and permit reusability, although the component-based approach (Ritter and Koedinger, 1997), (Roschelle and Kaput, 1995), (Suthers and Jones, 1997) has been advocated for reusing ITS components. However, ITS have not achieved either reuse or integration, and component reuse may be costly (Sparks et al., 1999), (Murray, 1996).

3.2.3 Requirements for a Generic Intelligent Architecture

This section describes the requirements for the generic architecture for ITS. The architecture is intended to exploit a platform in which a wide variety of ITS can be easily developed. Like most research efforts, the detailed design, implementation, and testing stages uncovered further requirements that were not apparent during the initial analysis. These requirements were derived

from several ITS research publications and they are implicit in software engineering literature. Furthermore, these requirements can be tied directly into an implementation.

These requirements were also drawn from earlier implementation and testing of the generic architecture (Atolagbe and Hlupic, 1996), although these tests produced no significant changes to the requirements, probably due to their relative simplicity compared to ITS architectures (e.g. (Eriksson et al., 1996), (Musen et al., 1995)). However, they resulted in a re-conceptualisation of the generic architecture (Atolagbe and Hlupic, 1997; 1998) to take advantage of constraints identified from earlier architecture with considerable semantics checks (prudent attribute evaluation).

The design considerations of the Generic Intelligent Tutoring System Architecture (GeNisa) were driven primarily by the following requirements:

i. *Design Architecture.* One of the objectives of the generic architectural is to exploit a system's structure, at different levels of abstraction; also, to define different components of the system and the interactions between them.
ii. *Consistency.* ITS components are developed by multiple rather than single software developers, and in the process they use various tools to produce different components (Murray, 1999). There is also an issue of consistency violation between the modules, which may hinder reuse. Therefore, the need arises to assist ITS developers in the production of different ITS tools that can eliminate consistency violation and support editing of multiple components modules.
iii. *Reducing the Developer's Effort.* The basic objective of this is to develop a framework under which one may easily build various components into the architecture. The generic architecture should make application development easier, reduce the efforts and time required for developing an ITS.
iv. *Open and Dynamic.* Facilities should exist that allow new components and systems to be integrated into the architecture. The architecture must be transparent and it must allow for easy modification, and it must be able to encompass changing data relatively easily and at low cost. Authoring should be easier if it takes less time to develop and reuses existing components. Since this objective emphasises generality and flexibility, it is also appropriate to choose examples of graphical, semi-formal, non-knowledge representation languages to generalise the applications. For example, Object Model Notation (Coad et al., 1995) was used for design of the GeNisa architecture.

v. *Complete and Autonomous.* Since this research is conceived as a framework to be used as the basis for other ITS systems development, the structure of its design must be very clear and unambiguous. All component events and behaviour should cover as many applications as possible. This requirement increases the chance that the component methods will be suitable across domains. This may facilitate adaptation, since component behaviour can be customised. Furthermore, separation of component events and methods may provide a logically precise design that can serve as a reference point to resolve design ambiguities (Gerhart et al., 1994).

vi. *Distribution.* The architecture should permit applications modules and other resources to be deployed over a wide area in which there is an access to resources. The system should not be restricted to a particular domain, and should be able to integrate information from local different repositories. The objective also exploits the use of class libraries for distributing applications and the use of *design patterns* (Gamma et al., 1995), (Pree, 1995) to provide detailed, semi-formal documentation of design concepts. The design pattern provides a vocabulary and starting point for design, and can be encoded as general-purpose classes in object-oriented class libraries (Nelson, 1995), (Stepanov and Lee, 1995).

vii. *Modularity.* The functionality of an ITS architecture should be distinct from other processes taking place within the architecture. Components interoperability should be domain-independent and should consist of modular components. Therefore, little explicit encoding of domain knowledge should be required from developers. Because little knowledge may be available, the techniques need to be robust enough to provide a developer with "useful" feedback during development. This requirement implicitly encompasses a clear separation between the architectural components, and the underlining events and the user interface.

viii. *Knowledge Acquisition.* A toolkit that allows developers to augment and modify knowledge-based systems automatically is required for acquiring knowledge from user activities during instruction. The system should gather input from both the student activities and from the instructional environment. The system should observe the user's activities as a sequence of action-examples that it receives from the knowledge bases.

ix. *Reusability.* To support the construction and maintenance of ITS from libraries of reusable existing components, and knowledge bases. The development language should not be specific to a domain, in order to support developers for different domains and across platforms.

x. *Heterogeneous Support.* GeNisa components should integrate seamlessly with other applications and provide means of sharing data and knowledge. The architecture should integrate different formalisms into a multi-paradigm representation and define the static semantics between components so that any component specification errors can be detected.

By using a programming language that meets these requirements, this research can demonstrate the feasibility of developing an ITS, by combining software engineering principles and AI techniques. Furthermore, these requirements are used to describe the different level of the system's granularity, and to encapsulate different parts of the system in separate, hierarchically structured representation.

3.2.4 Software Architecture

Software architecture is an approach for developing software systems (Soni et al., 1995), (Kruchten, 1995), (Shaw and Garlan, 1996). Software architectures are design representations of software systems that are based on composition of software components and their interaction (Shaw and Garlan, 1996). This approach provides ways for selecting and developing software components visually (Nelson, 1993). The component-based approach (Ritter and Koedinger, 1997), (Suthers and Jones, 1997) to software development is based on sharing *semantics* across components and applications. This represents a framework for understanding and representing components in the context of the system (Liebenau and Backhouse, 1990). Therefore, software architects should support developers in making design decisions, and on application components, together with their event processes.

It is commonly accepted that application components can be stored in a knowledge base of a particular type (Habermann and Notkin, 1986). The knowledge bases may have been created, accessed, updated and administered by using the functionality provided by a knowledge base system (Habermann and Notkin, 1986). The methodology presented in this thesis aims to evolve GeNisa into an integrated, adaptable and generic architecture with heterogeneous knowledge base support. The GeNisa requirements delineated in the preceding section is used to determine toolkit requirements for developers and for learners.

The premise of the generic architecture is based on providing access to collection of classes and tools, organised into hierarchies for abstracting into an application. The architecture consists of strategies for handling both intra and inter-component relationships. Components are therefore, required to provide users with the means to access and update the syntactic structure of tools incrementally (Engels et al., 1992) in order to facilitate the required components update and associated component development.

Having established interactivity related strategies in the preceding sections, and based on the literature, this was conceptualised into an architecture with a reusable design and instructional environment that could be used, extended, and integrated with ITS.

ITS research is generally moving towards the integration of a wide variety of software components developed by different research communities (Gaines, 1994). For example, requirements for integrating user-modelling tools have been suggested by Wenger (1987). This approach provides means for using adaptive instruction module in an ITS (Wenger, 1987).

It is widely recognised that class hierarchies in object-oriented programming languages are used for multiple purposes (Elson-Cook, 1990). The approach allows the separation of specification and implementation into separate hierarchies. The main advantage of this approach is that it allows the generic architecture to be represented in hierarchical class structure, which makes the architecture more flexible, and it can be suited for different needs.

This research also exploits the Model-View-Controller (MVC) architecture in order to provide an object-oriented framework for developing reusable, extensible objects by dividing the functionality of an application into three logical objects (i.e. model, view and controller) (Buschmann et al., 1996), (Cox, 1996). This approach divides a user interface into three sub-systems: model, view, and controller. A model is a fixed description and representation of the objects being modelled, while views represent different ways of "viewing" the objects. MVC assumes that all entities that will be modelled in the system can be anticipated. In the generic architecture framework, classes may be both models and views at the same time. Classes represent and present the objects created locally, and they may represent objects created in other classes. This allows the system to evolve the model and the views at any time by separation of data from presentation.

The generic architecture development environment implements the MVC design pattern in order to support multiple views (Krasner and Pope, 1988), (Gamma et al., 1995). This approach allows the combination of object view and controller roles into the same object and processes all the input and output of the software. Also, it is used to translate users' action into events (Gamma et al., 1995) by accepting input from the user, and instructs the object and views class to perform actions based on that input. This approach allows greater modularity, and allows the same design representation to be viewed and to be easily and efficiently updated to reflect current changes. Every component of the DesignObject (depicted in Figure 3.1) is an observer of the corresponding component according to the Observer pattern (Gamma et al., 1995). Layers

(discussed in Chapter 4) and DesignObjects act as models for the visual properties of the diagram. include co-ordinates, colours, etc. Underlying all components is a class library with modules that can be reused.

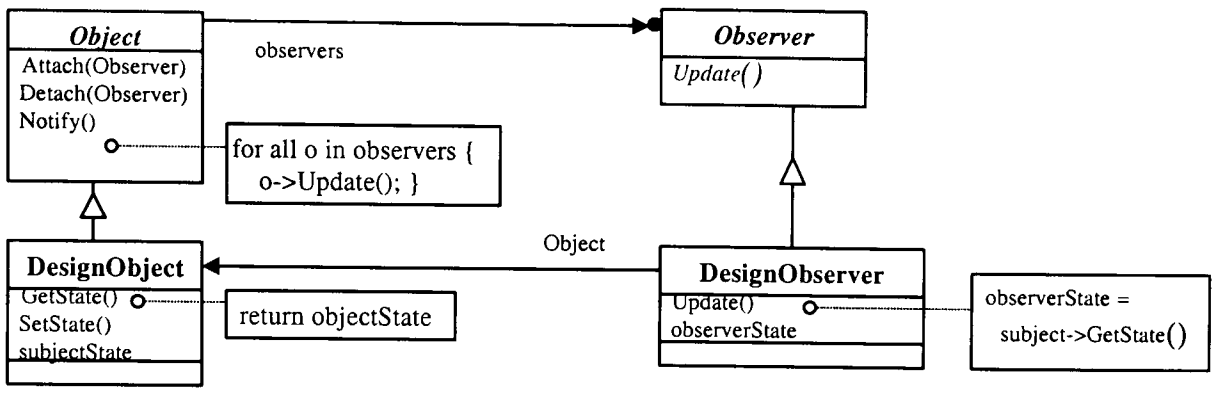


Figure 3.1 Observer Pattern (Gamma et al., 1995)

Most of the components used within a software development environment may rely on the events provided by other components. For example, a problem-solving component directing the activity of a user should have access to communication and execution of all related event modules if it is to operate effectively. However, the provision of these capabilities may be provided through an integrated environment.

3.2.5 Consistency Constraints

Different levels of details have to be considered for the specification of component consistency constraints and their validation (Rumbaugh et al., 1991). From a developer point of view, consistency constraints are concerned with investigation into which set of components a particular tool is consistent or inconsistent with. A component that is not consistent with other similar tools may require further enhancement. Hence the generic architecture must ensure that there are no consistency constraint violations during the lifecycle of the components modules and their events processes. This includes static semantic constraints of the formal languages and consistency constraints between different components. These constraints should not be confined to tools of the same type but between tools of different types. An important factor considered by this research is whether these constraints have been defined properly and are respected by tools developed during the research.

3.3 Methodological Assumptions

A wide spectrum of ITS development activities, ranging from the system design, conceptualisation, sharing, and reuse (Biggerstaff and Richter, 1989), (Rumbaugh et al., 1991), (Eriksson et al., 1996), (Musen et al., 1995) are necessary during ITS development. This section discusses the underlying assumptions for this research. These assumptions provide the context of which this research shall address the design and implementation of the generic architecture for ITS. They also (i) characterise the knowledge representation formalism in order to facilitate portability and reusability, and (ii) characterise the problem solving methods for appropriate interaction with the different components within the generic architecture. These assumptions can be used to develop new components, or to adapt existing ones. However, these activities can be integrated into a platform built upon a set of knowledge base toolkits for application development.

The following methodological assumptions were derived mainly from the literature, and they define all the properties of ITS component and their cross-platform needs. Also, it states the major assumptions made about the generic architecture environment.

i. ITS uses various instructional strategies for effective instruction (Murray, 1997), (Angelides and Paul, 1995), (Angelides and Tong, 1995), (Major, 1995). These strategies have been influenced by theories of learning and skill acquisition, and it is assumed that it is possible to identify instructional factors from the body of knowledge such as learning theories, cognitive psychology and instructional planning (Anderson, 1993), (Shute, 1995). The aim is that these theories will provide the basis on which the instructional module is based and a method that defines *knowledge roles*, which supports the different elements of the instructional module.

ii. *Reducing Development Lifecycle*. One of the main difficulties in designing ITS is the time and cost required (Reinhardt and Schewe, 1995), (Murray and Woolf, 1992). ITS development is often fragmented and less successful because of the multidisciplinary skills involved (Murray and Woolf, 1992). Reducing development time may be realised by promoting a degree of independence among the generic architecture components, and therefore offers the potential of being able to share and reuse these components.

iii. *ITS Components Modularity*. Simplifying ITS construction must take advantage of the modularity of each component of the system. The modularity of components should make easier transferring the tutoring to a new domain. The system's modularity should also allow the transfer of general-purpose components (for example inference engines,

and pedagogical agents) more easily. This should not involve developers simply reusing their own components, but should also mean sharing components among different ITS developers across platforms.

iv. *Authoring Tool.* The objective of authoring tools is to provide a development environment for the construction of ITS (Murray, 1999). There are two main approaches to achieving this goal: (i) to provide a simple development environment for developers to develop their own tools, and (ii) to provide an easier means for developers to represent the domain and design ITS components visually. This research investigates how to extend the capabilities afforded by these approaches, in order to preserve the level of usability and functionalities inherent in ITS components, and add additional components, features and authoring paradigms to allow more powerful and flexible instruction to be developed.

v. *Knowledge Reuse.* Many ITS researchers are seeking to reuse knowledge in new applications and to share encoded knowledge across different domains and platforms (Chandrasekaran et al., 1999), (Chandrasekaran, 1988), (Chandrasekaran, 1986), (McDermott, 1988), (Breuker and van de Velde, 1994). Knowledge reuse involves many dimensions, including the reuse of schemas, various software modules, ontologies, diagnosis and inferences, and problem-solving methods. Reuse of knowledge is based on the assumption that a generic intelligent architecture consists of multiple components developed from different Java classes and other development tools.

vi. *Knowledge-based Systems.* The representation of knowledge ensures that the knowledge-base contents are reusable and can easily be modified. This poses a requirement that knowledge-based systems must have the ability to use multiple schemas describing the contents of the same knowledge base.

vii. *Knowledge Acquisition and Reasoning.* The generic architecture learns by making decisions on when and what to learn (as well as determining if learned knowledge should be retained) during instruction. These decisions are made in order to maximise the utility of acquired knowledge and for automatic knowledge acquisition from the user activity.

viii. *Modularised Learning.* These methods allow the generic architecture to be used for exploring relationships between problem-solving and learning methods. A central assumption to the use of these modularised learning methods is that the user interface between different methods either does not exist or can be completely described (and thus anticipated).

ix. *Pedagogical Agent.* The pedagogical agent "knows" every part of the case scenario, and the domain knowledge associated with the scenario, and the correct problem solving

methods. It also assumed that the user interacts with the pedagogical agent during instruction.

x. Since application would be deployed over the World Wide Web, the following assumptions are made: (i) Application procedures are small and modular, i.e. components' procedures are broken down into sets of small modular procedures. These smaller units are used to develop large procedures. This assumption is used when considering the run-time overhead of some components and algorithms. (ii) Logically-related steps/procedures are grouped together.

These assumptions are consistent with current paradigms for developing ITS (Murray, 1999), (Suthers and Jones, 1997), (Musen et al., 1995), (Woolf, 1992) and provide rational justification for the ethos of the generic architecture.

3.3.1 Knowledge Sharing and Reuse

Different varieties of representational formalisms have been used for control and strategic knowledge in ITS shells (Murray, 1998), (Self, 1999). Some of these systems employ sophisticated AI techniques such as goal-based planning (Russell et al., 1988), black board architectures (Murray 1990), agents (Cheikes, 1995), task decomposition (Van Marcke, 1992), and production rules (Anderson and Pelletier, 1991), (Major and Reichgelt, 1991). Furthermore, no framework or visual component editor has been developed for any of these formalisms, which increases their usability. Although, some of these representation formalisms are modular, they may affect structure of knowledge, and make design decision complicated (Lesser, 1984).

The generic architecture uses the Unified Modelling Language (UML) (Object Management Group, 1998) paradigm for developing ITS. The justifications for this approach are: (i) UML is primarily graphical, with textual annotation. These notations can be used for developing ITS components; (ii) UML models include elements such as software components, communication mechanisms, processes, threads, components events, external systems, and source code modules (Garlan and Shaw, 1993), (Kruchten, 1995), (Luckham and Vera, 1995), (Taylor et al., 1996). These elements may be instantiated during application development into different components with different functionalities.

This research exploits the use of UML as architectural support tools for ITS. This approach provides support that is feasible because it provides a useful and extensible set of predefined constructs that can be extended, and it is based on more concise software engineering methods. This approach will allow developers to externalise their mental model of a given problem and

solution (Shneiderman, 1998) and allows the model to be analysed, and communicated for different purposes. For example, the expert model may be regarded as more advanced model components for instructional purposes. The design environment postulated in this research supports the ethos of the generic architecture by using:

i. Flexible data definitions for various components of the generic architecture. These data definitions will include standard definitions for knowledge databases, design components, inference modules, student model, and pedagogical module. These components are used for specifying *what* to teach, and teaching strategies that specifies *how* to teach (Wenger, 1987), (Ohlsson, 1987).
ii. Transaction definitions for interaction between the components of the generic architecture. This would include a definition of information needed and standard processes to be followed by the controlling module of the architecture.
iii. Strategies and communication standards for interaction between the architecture and other software, such as design tools, or other ITS software.

Nevertheless, the development of intelligent instruction from generic, reusable components could benefit from standardisation of ITS components, which may provide enhanced development tools, with improved interoperability between components. As ITS development involves hierarchical decomposition of domain subject (Murray, 1999), the UML which supports hierarchical decomposition maybe feasible for developing ITS.

3.3.2 Component Behaviour

ITS components are analysed as knowledge communication subsystems (Wenger, 1987). This approach allows instruction to be individualised; allows tutorial misconception to be diagnosed and to provide appropriate remediation (e.g. (Van Merrienboer and Krammer, 1992), (Anderson et al., 1990)) to the student. Therefore, pedagogical goals and sub-goals are used as the basis for providing appropriate diagnosis during instruction.

Murray (1998) argue that there is no distinct methods to determine the composition of an ITS component, because ITS components are bound into an implementation structure, which makes it more difficult to reuse or extend the components behaviour independently. Therefore, this research assumes that ITS component behaviour is based on the events within the GeNisa architecture. Moreover, any initiated components events are declarations about component

behaviour, and not domain specific behaviour. Therefore this research assumes that a component can be expressed as a combination of a range of component classes with different behaviour.

The component behaviour is encoded so as to provide dynamic feedback to the user during problem solving. It is based on cognitive model of learning through problem-solving and using case scenarios (VanLehn, 1992). This approach may facilitate real-time control of problem solving activities by the user, and assist in structuring the presentation of tutorial. Nevertheless, the component behaviour is specified by global constraints, which relates to the users' current task. This constraint is implemented by a software agent, which ensures that no components event is initialised unless it is invoked by the user. This mechanism provides support for the implementation of control protocols in the context of open and evolving dynamic environment.

3.4 Knowledge Representation

The contents of a knowledge-based system constantly change during its life cycle (Clancey, 1992), because of the changing needs of the users. Therefore, this section discusses knowledge representation in the context of the generic architecture, which relates to knowledge based systems analysis, design, reuse, and/or systematic decomposition of the domain subject (Murray, 1999), (Ramachandran and Fleischer, 1996), (Frakes, 1994), (Krueger, 1992), (Neighbors, 1984). It addresses the knowledge acquisition bottleneck (Murray, 1997), (Hoffman, 1987) by employing: (i) a representation of knowledge structures, (ii) systematic decomposition of the domain subject, (iii) a set of domain independent attributes, and (iv) a meta-knowledge of the domain subject.

Figure 3.2 Knowledge Representation

Figure 3.2 depicts a schematic representation of the knowledge representation methods. The *DomainRepresentation* class implements the domain representation formalism, which consists of *DomainObject* and an instance of the class derived from the abstract class *AbstractMap*. An abstraction map is used to store the relationship between an element's structure and the structure of its representation. The *DomainBindingsMap* class is a subclass of *AbstractMap* used to store a set of pairs of elements. The first element in each pair is a child of the element being represented and the second element is contained within the subsystem inside the representation. It is used to represent a simple mapping between "external", or more abstract, structure and "internal", or the less abstract structure within the representation.

The knowledge representation construct are organised into a structure of classes according to object-oriented programming methods (Booch, 1994). These classes support hierarchical decomposition of domain tasks and methods (Chandrasekaran et al., 1999), (Musen et al., 1995), (Steels, 1993), (Musen, 1992), as well as a hierarchic classification of domain subject. Moreover, synthesis of knowledge-based applications is explored by combining domain specific and domain independent components.

ITS use separate knowledge bases, for instructional content, and for teaching strategies (Murray, 1998), (Murray, 1996), (Clancey, 1987), (Wenger, 1987). However, the use of knowledge-based paradigm in ITS development must conform to traditional ITS architecture (Siemer and Angelides, 1998). This research uses a knowledge representation formalism that allows combination of different tools and other software modules to engineer a sound pedagogical, intelligent interactive instructional system. The core purpose of this approach is to support component-based development, modularity, and reuse of the generic architecture components.

3.4.1 Components and Interoperability

Objects are described as entities, which are characterised by their attributes and methods (Booch, 1994), (Rumbaugh et al., 1991). The component method defines its state and events processes, problem-solving method and also defines its interaction with other components (Musen et al., 1995).

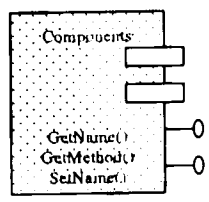


Figure 3.3 A Component Diagram

Components interactions are defined through their interfaces (Booch, 1994), (Rumbaugh et al., 1991). An example of a component diagram is shown in Figure 3.3.

This research investigates how ITS can be modularised as object-oriented software components. Such ITS components should encapsulate multiple classes, which can themselves be viewed as components classes, as they support the object-oriented mechanisms of encapsulation and inheritance (Booch, 1994). Components are often defined as pre-fabricated software components that can be combined in a 'plug-and-play' manner (Orfali et al., 1996). As such, components can be implemented as objects or as compositions of collaborating objects, and packaged as independent pieces of code.

An important characteristic of a component-based approach is that individual components may contain specialised functionality and/or knowledge and have access to the functionality of other components in the system. This has a couple of advantages for a generic architecture for ITS. First, this allows different components of the system to use the same event protocols during instruction. Second, it allows all components' properties and behaviours to be easily abstracted from the system by carefully defining their internal state and data structures through an external interface (Stead et al. 2000), (Musen, 1999), (Booch, 1994).

The components based approach has some architectural implications. For example, the different software components must conform to a common user interface look and feel (Nelson, 1993), (Gosling et al., 2000), or the components need to support the use of their functionality by other components. Moreover, the structure of the system will be implemented through the use of different components packages, thereby allowing each package to be easily modified as required. A package is the prototypical modularisation scheme, with well-defined static interfaces and dependencies (Rumbaugh et al., 1991). This approach allows further issues of portability and reusability, to be examined further in Chapter 6.

This research uses ontology (Perez and Benjamins, 1999), (Chandrasekaran et al., 1999) (Gruber, 1993), (Mizoguchi, 1993), to provide a basis for building different components of the generic architecture and to enhance knowledge sharing and reuse (Fridman et al., 1997), (Mizoguchi et al., 1995), (Mizoguchi et al., 1996). The ontology representation formalism used in this research should remain invariant over various knowledge bases and across a range of domain (Guarino, 1997), (Guarino and Giaretta, 1995). Moreover, the ontology contains an explicit and functional representation of components class definitions and its instances. It provides a "conceptualisation", which can be "shared" by multiple components interacting during an event processing protocol.

Using ontology for describing component methods and attributes provides two advantages for reusing ITS components. It provides means for specifying the different component and methods in the generic architecture, with common, sharable attributes and allows its knowledge bases and problem-solving methods to be conceptualised as hierarchically structured taxonomy levels (Benjamins et al., 1999), (Chandrasekaran et al., 1998), (Fensel et al., 1997), (Gruber, 1995). Therefore, it could be inferred that ontology collectively serves as a set of reusable components for building ITS.

The following examples illustrates the various way by which this research uses ontologies for developing ITS. Firstly, as ITS are knowledge-based tutors (Murray, 1996), therefore ontologies provides reusable constructs, which can be utilised in the design of the knowledge bases for ITS and to be used across different domains. This approach is similar to the GAMES methodology (Van Heijst, 1995), (Guarino, 1997). Moreover, as ontology can be used for conceptualisation, it provides means for conducting analysis, design and comparison of different component models. For example, Sim and Rennels, (1995) and Visser and BenchCapon, (1996). Secondly, an ontology may also be used to describe the structure, knowledge types, topic properties, entities, methods and behaviour of components in a system during ITS development. An illustration of this example includes CUE (Van Heijst and Schreiber, 1994), (Van Heijst, 1995), (Mizoguchi et al., 1996). Therefore, an ontology allows instructional strategies, student model components, diagnostic models, pedagogical tasks structure and interactivities with the student to be conceptualised, designed and implemented for use across different domains (e.g. (Fridman-Noy and Hafner, 1997), (Mizoguchi et al., 1996), (Murray, 1996)).

Figure 3.4 depicts the main modules of the GeNisa conceptualised in this research. The architecture consists of four subsystems: the event manager that implements the interaction between components, and implements protocol of message passing between application; the user interface provides direct manipulation of the components (Shneiderman, 1998), (Nelson, 1993); the components modules are units of software modules with independent functionality (Ritter and Koedinger, 1997); and, finally, the reasoning subsystems function as reasoning "engines" of the system (executing a task may imply a call for reasoning service via the events manager). Each component may comprise of an arbitrary number of processes. Object-orientated approach (Booch, 1994) was adapted to provide high-level support for both inheritance and message passing. This approach enables this collection of autonomous processes to operate concurrently within the development/instructional environments, and to be represented in class libraries thereby allowing the functionality of the components to be extended.

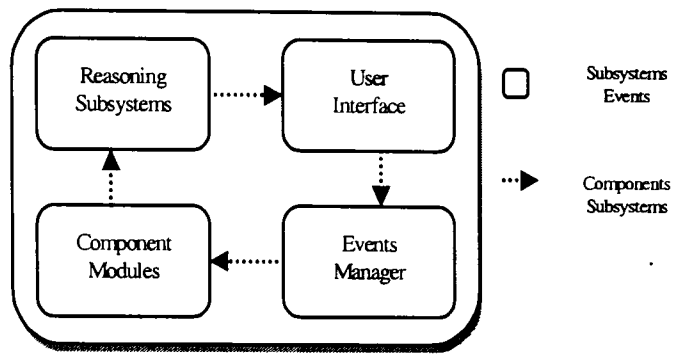


Figure 3.4 Main Modules of the Generic Architecture

Some of the properties of the GeNisa components include: (i) *Autonomy*. Component implementation functions are autonomous and may depend on inputs and output from their respective users; (ii) *Communication ability*. Tools should be able to interact with other systems in order to distribute tasks for solving a problem; (iii) *Reactivity*. Components should response quickly to user's requests and obtain feedback if required. This depends on the control strategy to be applied to the component, in order to accomplish the function; (iv) *Reuse*. Effectiveness and ease of use of developed components; (v) *Robustness*. Exceptional situations should be handled intelligently.

Taking these general traits into account, these properties may be represented as component features, which are properties of applications where the component might be used. This framework allows heterogeneous group of tools to interact (Johnson and Rickel, 1999), (Johnson et al., 1998), (Ritter and Koedinger, 1997; 1996), (Ritter and Blessing, 1998). The advantage of this approach is that it provides a model of the component properties of the generic architecture and the basis of interaction among different components. This approach may enhance flexibility and reuse by allowing components to be developed independently, used on other platforms and across domains.

3.4.2 Knowledge Base

This section outlines the possible roles of a knowledge base for use in GeNisa and in coping with some of the sharing and reuse problems and discusses the integration of knowledge bases and editing tools into an application development environment.

This research uses ontologies to annotate problem-solving methods and for problem representation (Mizoguchi et al., 1995), (Musen et al., 1995), (Fensel et al., 1997), (Benjamins et

al., 1999), (Chandrasekaran et al., 1998). Knowledge modelling methodologies such as KADS (Knowledge Analysis and Design Structure) (Wielinga et al., 1992), (Breuker, 1990), (Wielinga and Breuker, 1990), PROTEGE (Puerta, 1992), and MULTIS (Mizoguchi et al., 1995) originated from the generic tasks (Chandrasekaran, 1986) and heuristic classification (Clancey, 1985). Currently, knowledge engineering research are directed towards task ontology, which uses different concepts as building blocks for knowledge-based systems (Musen et al., 1995). Hence, an ontology provide a knowledge level description (Newell, 1982) that is independent of any representational formalism (Chandrasekaran et al., 1999), (Mizoguchi et al., 1996).

Task ontology (Mizoguchi et al., 1996), (Van Marcke, 1995) provides an effective methodology and vocabulary for both analysing and synthesising knowledge-based systems (Van Marcke, 1995). Moreover, task ontology can be use as a means for overcoming the shortcomings in current ITS (Mizoguchi et al., 1996) and for characterising and formalising ITS in order to facilitate reuse.

Building knowledge systems for use in an ITS involves creating a model of a particular domain such as simulation modelling, botany, or mathematics (Murray, 1996). This approach requires a conceptual shift from traditional "story board" representations of tutorial content to more modular knowledge based representations (Murray, 1996), (Murray, 1999). Such a model is usually an abstraction of the domain under consideration (Wenger, 1987), (Murray, 1996). Moreover, this model is concerned with which elements of the domain should be modelled by the system. Therefore, ontology is used as specification of domain knowledge, used as building blocks for developing an ITS, and it provides reusable components for the generic architecture (Sim and Rennels, 1995), (Studer et al., 1999). The framework for the specification of the generic architecture consists of three main elements depicted in Figure 3.5. The framework consists of:

(i) *Domain knowledge*, which describes the structure on the domain, instructional strategies and the problem solving methods (Top and Akkermans, 1994). It involves the characterisation of the case scenario, domain knowledge, problem-solving strategies (for diagnosing users activities) and the pedagogical task structure. Each of these elements of the domain knowledge are described independently in order to foster reusability across different domains (Breuker and van de Velde, 1994), (Top and Akkermans, 1994), (Puppe, 1993) and to support schemas integration.

(ii) *Diagnostics*, which involves description of the diagnostic processes that can support different the users' differences from different sources (including knowledge bases, and over the Internet).

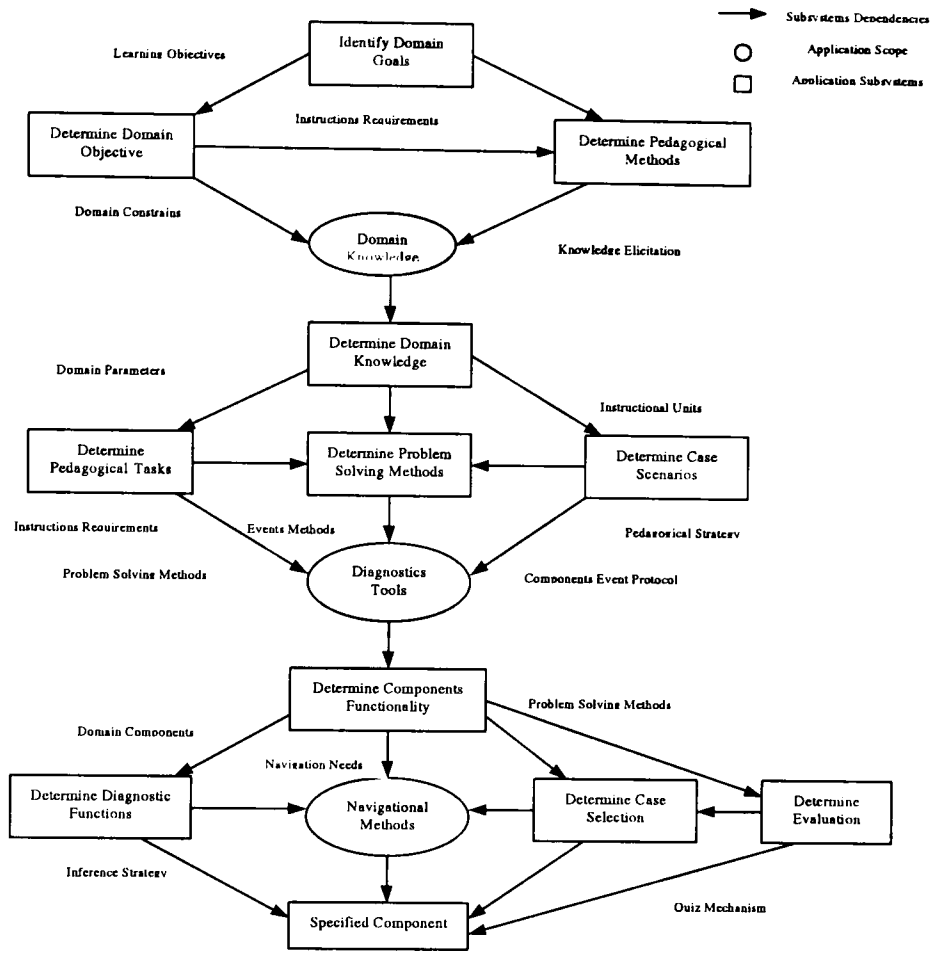


Figure 3.5 Instructional Design Process

This involves a description of the inference mechanisms, and diagnostic methods for querying users' activities. Diagnostic description may support the use of variety of strategies to aid the student learning. This may requires the use of temporal logic (Treur, 1994) in order to specify the dynamic interaction between the user and the system. Finally, the navigation, which describes the ability to present instruction dynamically across heterogeneous platforms (Weber and Specht, 1997). Navigational methods support different pedagogical techniques and instructional sequencing, and support navigation over the Internet. It supports adaptive navigation with an individualised user model (Weber and Specht, 1997). The navigational methods can guide the development of tutorial contents and how to sequence the contents. It provides navigation and information retrieval features, and information for managing the content of instruction across different domains. This is important because knowledge of the subject matter relates to an ability to navigate in a non-linear way through the tutorial space (Ohlsson, 1992). Each tutorial content is linked by using contents "links", which can be encapsulated in the knowledge-based. Methods such as "Submit", "Elicit", "Concepts" (describes tutorial constraints), "Remove_Tutor", "Insert_New_Tutor", are used to describe and modify the structure of knowledge bases and to characterise tutorial navigation methods, which are relatively independent of other components in

the generic architecture. This may facilitate presentation of tutorials irrespective of the semantic structure of the knowledge bases.

The knowledge base can be considered as an open knowledge based system with different levels of conceptualisation (Chandrasekaran et al., 1999), (Studer et al., 1998), (Guarino, 1997), and is classified into different levels of hierarchical structure (Wielinga et al., 1997). The main classification levels used in this research are:

i. *Domain Task.* Domain knowledge defines objects and relations that express a domain task-specific perspective on the domain knowledge, i.e. specification of a domain goal together with some input and required output (Studer et al., 1999), (Studer et al., 1998) and the reasoning behaviour of the domain. The domain task describes an ontology for a tutorial task, for example, mathematics, which contains objects, such as lesson objectives, instructional strategies, etc. Ontologies provide means of reusing and constructing the knowledge bases by the definition of knowledge and data (Chandrasekaran et al., 1998). This allows knowledge bases to be used as building blocks for designing, sharing, and reuse of the knowledge base development for instructional purposes (Murray, 1999), (Eriksson et al., 1996), (Musen et al., 1995). Furthermore, an ontology can be used to define objects and relations that express a method-specific perspective on the domain knowledge (a method means a specification of how a task can be performed or a problem solving methods (Musen et al., 1995), (Studer et al., 1999). This ontology contains methodologies, which specify various tasks, and contain objects, and solutions to some problems, constraints, and value assessment.

ii. *Knowledge Sharing.* Ontologies can be used as separate specification element that supports the reusable specification of data structures and to support the combination, adaptation and distributed execution of knowledge components from different components libraries (Fensel et al., 1997), (Benjamins et al., 1998). Part of the constituents of the generic architecture is the ability to share semantics amongst different components; therefore, it is essential that a shared semantics is achieved. One use of these semantics is for mapping the component's internal representation to a task that the user wants. Another use is building default descriptions of a component that could be used with different users. The sharing of semantic contents may be achieved by defining common ontologies for the generic architecture, and using clearly defined mappings for each ontology of the system's components. This approach helped to define an ontology allowing each system to have its own ontological representation, for sharing task knowledge (Eriksson et al., 1995) and component functions.

The sharing of the individual ontologies requires mappings between the knowledge base and problem-solving method (Eriksson et al., 1996). This approach allows the mapping of ontologies to be implemented by specifying in advance set components, which provides a meta level method for controlling performance and for making mapping decisions (Fensel et al., 1997).

The knowledge base contains the representation of the subject matter and is suitable for the integration of intelligent tutoring and hypermedia systems (Murray, 1999), (Murray, 1998), (Angelides, 1998), (Merrill, 1983), (Merrill, 1996), (Wenger, 1987) because its structure is supposed to support the selection, sequencing, and presentation tasks of users' activity. This approach enables multiple navigational communication between the student model (Brusilovsky, 1998), (Brusilovsky, 1996), (VanLehn, 1996b), (Woolf and Murray, 1994), (Ohlsson, 1986), and it makes possible to access large quantities of tutorial material in a flexible and interactive way.

An object orientation representation scheme was used to implement various components in the knowledge base. In this representation scheme, domain task is represented as an object, with a list of attributes. View integration is used during conceptual design to produce a global description of the component on the basis of multiple descriptions of each class. View integration reflects a planned integration during component design, where different design problems are integrated into one (Booch, 1994). This approach allows the same component properties to be represented by different constructs in different component versions. For instance, the same component properties may be represented in terms of an entity/class as well as an attribute. To ensure the knowledge base behaves consistently with other components, this research assumes a global perspective for all components schema. In this way it is possible to detect all dependencies that exist between component of the architecture. This framework emphasises components modularisation and reusability (Booch, 1994), (Rumbaugh et al., 1991), which constitutes a unified representational formalism.

Most research on knowledge representation methods (Park et al., 1987) has concentrated on the problem-solving domains (Wenger, 1987), (Benjamins and Fensel, 1998). In this research, more emphasis is placed on developing "knowledge" of expertise for problem-solving tasks by using case scenario. This approach is different from the task analysis (Durkin, 1994) methods of instructional systems development. First, this approach can be implemented to yield a "simulation" of the expert behaviour. Second, it emphasises the order in which domain tasks must be performed as opposed to the order in which they must be learned (Johnson, et al. 1998), (Gagne and Briggs, 1979). Furthermore, this approach considers the notion of multiple views of the curriculum. According to Lesgold (1988), in some domains it is possible to view the

curriculum through different perspectives, which thereby allows combining content knowledge from multiple sources. For example, a basic simulation modelling course can be viewed through four different perspectives: simulation methodology, simulation software, design and analysis of experiments, and output analysis (Atolagbe and Hlupic 1997).

Each tutorial task consists of attributes, which are used for tutorial presentation and for case retrieval from the knowledge base. Each tutorial task represents an atomic concept in the knowledge base and can be directly associated with content features. The matching formalism is based on using Bayesian network inference methods to associate instructional task features and associated case scenarios. The ability to represent any given level of knowledge within the domain by using Bayesian methods with known properties could make the knowledge representation methods more interactive for use with the pedagogy module. For instance, Bayesian network has been used for automating medical diagnoses and supporting expert decision-making (e.g. (Haddawy et al., 1994), (Heckerman, et al., 1992)). Moreover, the Microsoft Office Assistant employed Bayesian network to infer a user's requirements by considering a user's background, interactivities and queries (Horvitz et al., 1998). Similar approaches have been used for assessing student ability during instruction (Martin and VanLehn, 1995), (Gitomer et al., 1995). Bayesian networks allow the knowledge-representation formalism to reduce the degree of complexities and accuracy can be varied, allowing content-sensitive information to be incorporated into the tutorial (Heckerman, 1997).

3.4.3 Instructional Planning

The instructional planning (Macmillan et al., 1988) contains specifications for different instructional methods, including how to present instructional material, what type of questions to ask, and when to intervene (Major et al., 1997), (Major, 1995), (Wenger, 1987), (Ohlsson, 1987). The instructional strategies consist of the main components for effective instruction (Dick and Carey, 1996). Self (1994) asserted that most of the currently existing ITS have little control over how knowledge gets presented to the student. This may be attributed to the fact that some ITS have been designed around a paradigm where instructional interactions are based on diagnostic actions by the learner (Vassileva, 1995). Furthermore, a tutorial planner must be able to generate tutorial plans, monitor the execution of the plans, and generate new plans (Eliot and Woolf, 1995), (Van Marcke, 1992). It must be able to adapt its plan when necessary in order to customise tutoring plans for each student (Eliot and Woolf, 1995), (Katz et al., 1993), (Katz and Lesgold, 1993).

Essentially, the instructional planning should allow developers to reuse the content of an existing courseware package and to tailor the teaching strategies to different users needs. This approach requires the developer to provide data for use in a predefined instructional module. This research adopted Goal Based Scenarios (GBS) (Schank, 1994), (Bell, 1998) as instructional strategy. Essentially, GBS based approach involves instructional planning and allows students to work on different scenarios using a predetermined framework (Bell, 1998). The GBS framework allows the developer to provide case scenarios, tools, graphics illustrations, hints, questions and answers, to be integrated to the instructional environment. Learner participation during instruction was emphasised through its hands-on approach. The system ensures that users participate in all aspects of the course and are presented with appropriate case scenarios. This strategy provides the learner with the skills necessary to meet the pedagogical objectives of the unit (Gange, 1985). Furthermore, this approach allows the tutor to share control during the tutorial session with the student and also allows the planner to manage the tutorial dialogue (Major, 1993).

In the context of this research, instruction is defined as a form of *structured learning experience* (Murray, 1996), (Gruber 1987), or as a systematic activity that is aimed at learning and includes teaching (Murray, 1999), (Pontecorvo, 1993). Success in applying AI-based instructional planning systems to ITS requires an interactive and automated method. An interactive method (to aid the user in manipulating knowledge), and automated methods (to verify the interactivity methods and extract new knowledge from a variety of sources) are needed, including on-line databases, training exercises, and actual tasks. Instructional planning is concerned with delivery and content planning (Wasson, 1996), (Wasson, 1992). Pedagogical decision making is concerned with both the content (what goals to focus on) and the delivery (how to achieve the goals) of instruction (Wasson, 1992), (Dijkstra et al., 1992). This research adapted the following instructional strategies:

i. *Content Planning*. Content planning entails generating and structuring instruction according to pedagogical classification tasks and the formation of a new plan as the student progresses (Wasson, 1996), (Wasson, 1992). The design of the content planning module should consider the following strategies. (i) Student mental model (Self, 1999), (Self, 1987), (Anderson, 1987), and knowledge requirements at different levels of instruction. This is essential in order to provide coherence and continuity in the instructional process (Wasson, 1996), (ii) Courseware material will be content specific; identify commonality between topics and extract new topic (Bruner, 1990), (iii) Range of tasks must be relevant to current operation and tasks should be annotated with appropriate case scenarios.

ii. *Delivery Planning*. Delivery planning is concerned with choosing the activities and instructional interactions that help the learner achieve the goals (Wasson, 1996), (Merrill et al., 1992). Delivery planning determines how to sequence explanations, tests, presentation, exploration, and how to manage the initiative in a tutorial dialogue. KAFITS (Murray and Woolf, 1992) uses a four-layered representation of instructional primitives: goals, topics, presentations and responses, and a template network representing a "dummy" teaching strategy. The developer can see it as a general delivery plan, which can be instantiated.

iii. *Hierarchical Planning*. Hierarchical planning is used to distinguish between goals and actions based on degrees of importance, so that the most important problem is solved first (Yank, 1997). In hierarchical planning, the domain knowledge is divided into different abstraction levels, so that a complex task can be structured into a set of ordered tasks and subtasks (Yang, 1997).

These levels of planning "cannot in practice be so clearly separated" (Murray, 1988). However, instructional planners can be represented as structured or partially structured goals, with different levels of granularity. In this research, instructional planning adapted some of the above strategies plus the following: (i) Default setting for novice user that will adapt to users needs. This value is assumed at the beginning of the course except if stipulated by the developer. (ii) Providing remedial information during instruction (Ohlsson, 1991). (iii) Tasks monitoring, the planner can monitor the users' problem-solving activities and can dynamically adapt its plan in order to achieve tutorial goals. These features are discussed further in Chapter 4.

3.4.4 Discourse Planner

The discourse planner implements the tutorial planning (Woolf and Cunningham, 1987) and provides a mechanism for pedagogical decision making (Murray, 1988), (Woolf, and Cunningham, 1987), and for controlling interactivities between the learner and the system. It involves both tutorial content and delivery planning (Wasson, 1996), (Wasson, 1992), (Dijkstia et al., 1992). This planning mechanism divides the decision making process into different hierarchically organised levels. Each level successively refines the decision making process into a form such that a customised tutoring plan is generated for the student. This may facilitate understanding of the student's problem solving methods, which may enable improved diagnosis and selection of appropriate tutorial strategy. The discourse planner and the student models use a network-based representation that includes abstract concepts and relationships as well as

strategies for problem solving. This framework is designed to be extensible and flexible through the effective use of object-oriented principles and Bayesian network.

Discourse planning deals with "planning communicative actions between the tutor and the student within a lesson" (Murray, 1988). The discourse module uses a content planning mechanism to control interaction with students by extracting information from the student's problem solving activities.

This research employs a Bayesian network for case retrieval by using Bayesian reasoning strategies during student's problem solving tasks (Johnson et al., 1998), (Martin and VanLehn, 1993). Bayesian networks have been used to integrate student current activities with prior experiences and to update their problem solving activities via their instructional methods.

The discourse planner is responsible for selecting/or generating instructional goals, deciding how to teach the selected goals, monitoring the student's behaviour, and determining what to do next at each point during a tutoring session. The discourse planner makes two different types of decisions during the tutoring session: decisions about the content of the lesson and decisions about its presentation strategy. Although the early ITS largely focused on the delivery strategy of the planner, some recent planning research shows the integration of both aspects in building the planner (Murray, 1990). The discourse planner in GeNisa carries out both functions, since it is beneficial for the learner if the system can provide a global lesson plan. The lesson planner must update the tutorial goals dynamically as the student model changes (Wool, 1984), (Russell et al., 1988). Therefore, determination of an appropriate level of remediation during instruction depends on current tutorial level and the use of an appropriate planning strategy. The discourse planner considers factors such as the learner's current tasks, tutorial levels and type of feedback a student prefers before remediation.

3.4.5 Teaching with Case Scenarios

Learner can acquire new skills by learning from examples (LeFevre and Dixon, 1986), (Pirolli and Anderson, 1985), (VanLehn, 1986). However, the benefit of learning from examples strongly depends on the student's study strategy, self-explanation, and the student's problem solving methods (Chi, 2000), (VanLehn, 1996a), (Ferguson-Hessler and de Jong, 1990), (Pressley et al., 1992), (Chi et al., 1994), (Renkl et al., 1998), (Pirolli and Recker, 1994). Using self-explanation strategy is usually a more constructive learning process because learners are generating appropriate questions during problem solving and answering them for themselves (VanLehn,

1996a). This approach allows the learner to elaborate on their existing knowledge, and to use an appropriate problem-solving strategy.

This research has investigated the benefits of integrating case-based ITS that allows the student to use self-explanation strategies during problem solving (Bielaczyc et al., 1995), (Ryan, 1996), (Chi et al., 1994). Such a system must be able to monitor the students' problem-solving methods (Katz et al., 1993) and to generate explanations that can improve the students' understanding. One primary way to support learning by using self-explanation strategy is to engineer components, which can enable the student to explore domain knowledge unintrusively. This approach can facilitate the development of cognitive and procedural skills (VanLehn, 1996a).

Learning from examples is a necessary part of ITS because: (i) in pedagogical terms, it allows students to participate in controlling some real world scenarios (Schank, 1994), (Bell, 1998); (ii) it facilitates the development of procedural knowledge, which can only be gained by involving the learning by doing (Anderson, 1983). Procedural knowledge involves manipulation of real-world object (for example, operating a machine) also including knowing how to adapt procedures to a given situation. Since students must learn procedural knowledge by doing, then it seems feasible to allow learners to practice on a real world scenario.

Case-based reasoning (Kolodner, 1993) combines a cognitive model describing how experienced users use and reason from past experience with a methodology for implementing such experience. It provides a distinct paradigm for presenting "real world" scenarios during instruction. Case-based reasoning provides a conceptual framework in which to store case scenarios to be used during instruction.

Kolodner (1993) stated that a case is a contextualised piece of knowledge representing instruction for a domain task. This approach allows students to acquire the skills of an experienced user in a complex scenario, and minimises cognitive strain during instruction (Anderson, 1989), (Zsambok and Klein, 1997), (Schank, 1982). The approach could encourage metacognitive activities such as reflection and self-evaluation.

Schank (1994) suggests a case-based approach to learning. He postulates a "goal based scenario" that can be used as a framework for the learning environment and provide the scenario context that models real-world applications. A case scenario might have multiple perspectives and this formalism can be organised hierarchically. This research explores the feasibility of integrating case-based reasoning components pedagogically into the generic architecture. Each case scenario

can be decomposed into separate tasks, and augmented with content constraints (Kolodner, 1993) to allow easy presentation and adaptation of cases during instruction. Furthermore, Case-based reasoning has been applied to the tasks of explanation, planning, diagnosis, and tutoring (Fowler et al., 1995), (Harris and Cook, 1998). Where a case for an instructional system might describe a scene or set of events, a Case-based planner's case is likely to be a set of instructions for achieving some goal (Fowler et al., 1995).

VanLenh et al., (1992) asserted that learners who are capable of generating appropriate questions and answering them for themselves seem to learn more. This implies that learners can be considered as having a set of cognitive processes, which are used during problem solving and for generating self-explanations. A computer based approach has been used to generate explanation and as an instructional aid (Clancey, 1990), (Moore, 1996), (Moore et al., 1996), (Woltz et al., 1990), which may support collaborative tutoring (Ploetzner and Fehse, 1998). The implementation of the case scenario used during instruction is described in Chapter 4.

3.5 Intelligent Tutoring System Architecture

The previous sections discussed why a Case-Based-Reasoning (CBR) and knowledge base are required for generic architecture. The following section provides details of architecture for use on modular components developed in this research named GeNisa: Generic Intelligent Architecture for Instructional Systems.

GeNisa's architecture is shown in Figure 3.6 GeNisa and consists of the following main components: the presentation system, the domain expert, the inference engine, the pedagogical model, the student model, the knowledge base, the scenario library and the knowledge-base.

3.5.1 Presentation System

The presentation system in GeNisa is represented as a graphical user interface, which is multimodal. It consists of a Java-based interface for designing and editing, and for describing components sharing functionalities between the generic architecture, the users and various components. Furthermore, the user interface is used to provide intelligent assistance to learners during instruction, and to enable more interactions between different components of the system and the user. It defines major components of the user interface data to be displayed in each part, presentation methods, layout of the components, and behaviour for the components on the

interface. It allows developers to manage the information to be presented, with some indication about the features of the information that should be displayed. This research uses Java applets to provide all the primitives for the user interface and enables facilities for presentation of case scenarios and courseware materials. The interface supports the execution of the byte code of the applet and a programming interface to its internal functionality. The provision of adaptive user interface requires adaptive mechanisms to support different needs of the users (Dieterich et al., 1993). Essentially, the requirements for the generic architecture user interface are: (i) a unique interface is required for application development and for instructional delivery, (this will allow the user to use suitable user interface components as required); (ii) flexibility of the user interface control mechanisms, in order to cater for different levels of users.

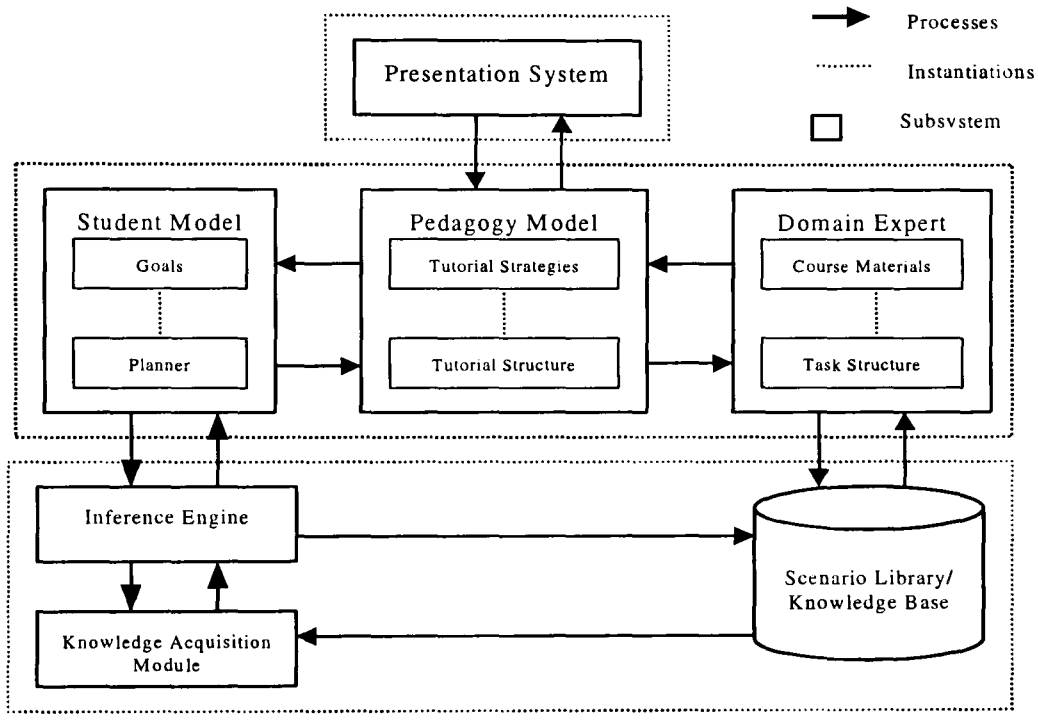


Figure 3. 6 GeNisa Architecture

3.5.2 Domain Expert

The domain expert is generally characterised by declarative/procedural knowledge (e.g., (Anderson, 1993), (Lesgold, 1988), (Shute, 1995)). However, the domain expert may be incomplete and therefore limited to instructing pedagogical tasks effectively (Anderson, 1988), (Lesgold et al., 1989), (Breuker 1988). Instruction requires both domain and pedagogy expertise (Anderson, 1988), (Lesgold et al., 1989). The effectiveness of the domain expert may be affected by the knowledge representation methods. Therefore, the domain expert should be represented by flexible knowledge representation formalisms, which are capable of effectively representing different range of knowledge and skills.

The domain expert contains "the specification of the goal structure that guides the teaching of a body of expertise" (Lesgold et al., 1989). The domain expert characterises the knowledge and strategies needed for expert performance in a domain (Murray, 1999), (Murray, 1997). It consists of the following types of knowledge: (i) support knowledge, which represents the general model of a domain (Clancey, 1986), (ii) operational knowledge, which represents the problem-solving procedure, and (iii) task specific knowledge. These knowledge types may improve interaction with the learner during instruction. Furthermore, the domain expert is used for remediating pedagogical task knowledge and for diagnosing learner's actions during instruction (Clancey, 1987), (Ohlsson, 1991).

The representation of domain knowledge must support different and correct diagnoses of the learner and should contain a description of its domain (Clancey, 1986). The student model uses the domain knowledge to build the student model (Self, 1999), (Self, 1990), (Wenger, 1987). The domain knowledge individualises the student model with problem-specific information when the learner selects a problem (Self, 1999). The domain knowledge also initialises the predetermined instructional material and appropriate case scenario to be elicited during instruction. It loads the problem-specific and case-specific information, and inputs cases and outputs strategies from the problem knowledge base into the learning environment.

In general, a domain expert "describes what is known about the world", and is used to provide a "description of some situation in the world." (Clancey, 1986). This description forms the basis for representing the domain knowledge in term of domain ontology and problem-solving structure (Van Heijst et al., 1997), (Chandreskaran et al., 1998) and may be used to specify the knowledge types, tutorial tasks properties, and link types (Murray, 1996). In addition, the domain expert has the ability to validate learners' problem-solving methods, compare them with pre-stored ideal solutions, and to dynamically generate appropriate instruction from the user solution. Therefore, the domain knowledge is used by the pedagogy expert to provide remediation for problem-solving during instruction.

This framework is different from using an overlay-model only because the overlay model (Ohlsson, 1993), (VanLehn 1988) assumes that the student's knowledge is a subset of the expert knowledge (VanLehn, 1988). However, a learner during instruction may employ a tutorial strategy that is not represented in the expert knowledge. Furthermore, overlay models do not address the situation where the student misunderstands the tutorial contents; they assume that tutorial content is complete or incomplete (Wenger, 1987). The domain expert can be used to provide dynamic control of instruction based on specific difficulties encountered by the student

during instruction (Shute, 1994), (Silverman, 1992), (Clancey and Soloway, 1990), (Ohlsson, 1991), and to guide knowledge elicitation.

3.5.3 Inference Engine

The contents of the knowledge base should support all logical constructs that may be needed, in order to perform a domain related inference operation. This can be achieved by employing an inference strategy that can invoke the appropriate inference object.

The inference engine is the supervisor for the various ITS modules during the instructional phase. During instruction, the inference engine consults the student model to determine an area of training that the student needs, and that is close to knowledge the student already has demonstrated. Next, the inference engine is consulted to determine the appropriate training event and instructional progression for the material. Based upon the training event, the inference engine may present a case scenario from the scenario library to the student (through the case elicitation), or may present some other instructional control module (from the expert module). However, it is essential to structure the knowledge base with respect to the functionalities of the inference engine, in order to satisfy requirements on efficiency and interactivity. The inference engine uses Bayesian networks (Pearl, 1988).

Bayesian networks provides an efficient computational techniques and flexibility for representing probabilistic dependencies, and has been used for inference, decision making, and problem solving in situations involving uncertainty (Conati and VanLehn, 1996), (Martin and VanLehn, 1995), (Pearl, 1988a).

A Bayesian belief network is a directed, acyclic graph (Pearl, 1988a). Nodes in a typical Bayesian network represent degrees of belief about a particular aspect of a domain, and edges in the graph represent causal dependencies (Pearl, 1988b). Degrees of belief are computed directly from prior probabilities using Bayesian inference. Implementation details of Bayesian network are provided in Appendix B.

The Bayesian network was also used to define how particular beliefs are propagated through the network through the use of probability tables. The network stores a table of probabilities for all possible combinations for the evidence nodes. When one probability value changes, the change is propagated through the network using the probability table at each node.

3.5.4 Pedagogy Model

The pedagogy model governs the structure of the training by presenting a variety of information from the different modules to determine the next training action dynamically. The pedagogy model embodies domain-independent pedagogical information. The primary purpose of the tutor module is to provide a training structure. This information is based upon Gagne's (1992) instructional events. The instantiation of the training strategy is accomplished through the use of domain-specific knowledge about concept complexity and relationships. For example, one part of the pedagogical strategy is to teach simpler concepts before more complex concepts. An example of this concept in the simulation-modelling domain might be to teach a simulation methodology before simulation model development.

Tutorial explanations and the use of case scenarios as tutorial strategies requires the student to continuously interact with the pedagogy model. Explanations and case scenarios may vary in the degree of cognitive processing they require of the student. The pedagogy module contains multiple strategies. The tutor model selects a strategy based on the current ontology/domain and the progress of the student. Current ITS literature suggests a variety of strategies for improving learners performance during instruction, for example (Anderson, 1990), (Ohlsson, 1993). Based on the literature review, the following describes the six instructional strategies that constitute the generic architecture pedagogy module.

i. *Learning with Scenarios.* This strategy uses a real-world scenario as the vehicle for instruction. Presentation of a scenario should involve two processes: demonstrating the scenario and teaching the correct operation in the situation. In demonstration mode, the tutor can present correct operational activities for the scenario.

ii. *Learning by Doing.* In this strategy the tutor is very active. Within the context of the scenario, it teaches the student step-by-step to perform the appropriate activity. At each step, the student can inquire about the purpose of the actions and activities performed.

iii. *Practising with Contents Feedback.* The tutor is less active in this strategy. The student performs activities without prompting by the tutor. When the tutor detects an error or missed action, it provides immediate remediation of the problem.

iv. *Hint.* The user in a problem-solving situation can benefit from advice or a hint from the tutor who continually watches the tasks and can correct the actions with in-depth explanations. Various types of guidance (on demand, automatic, with multiple explanations) can be obtained.

v. *Free Exploration.* The user can navigate into a case scenario that reacts to his actions without intervention or guidance of the system. The learner controls the activity (for example, the resolution of a task) and this mode can be compared with free navigation within a hypermedia document.

The overlay model is certainly not the most comprehensive model for representing adaptive instruction, but it is a well-established and adequate model (Ohlsson, 1993), (VanLehn, 1988). However, "the effectiveness of tutoring cannot be attributed to the implementation of a sophisticated pedagogy" (Graessern, 1993). Furthermore, the learning activities are initiated by the expert module, which can limit interactive learning activities, by the learner.

3.5.5 Student Model

The student model represents an estimate of the student's learning behaviour (Self, 1999), (Self, 1987), (VanLehn, 1988). One of the fundamental difficulties in representing a student model is the inability to represent and reason with uncertain information (estimate of the student's learning behaviour (Self, 1999), (Self, 1987), (VanLehn, 1999)). The GeNisa's student model uses a Bayesian network (described in Chapter 4, and implementation details are provided in Appendix B) for inserting new cases during problem solving and for model reformation. This research used a Bayesian network to represent the student model for the following reasons: (i) a novice learner would prefer learning with a case-based ITS than other systems because this may increase the system's ability to engage and motivate students, and (ii) multiple learning strategies can be represented within the model.

Uncertainty in student modelling has recently been addressed using Bayesian techniques (Cristina et al., 1997), (Villano, 1992). Uncertainty is unavoidable in student models, where instructional objectives and other information sources are unreliable, and the student's behaviour is unpredictable. The ability to reuse student model components can enhance the flexibility and the reuse of ITS in multiple contexts (Sparks et al., 1999).

3.6 Knowledge Acquisition

This research considers knowledge acquisition as modelling activity, where an abstract model is selected or developed which is then instantiated with application specific knowledge (Wielinga et al., 1993), (Ford et al., 1993), (Gaines et al., 1992), (Breuker and Wielinga, 1989), (Clancey, 1989). The approaches may help facilitate ways to characterise and organise the knowledge

acquired from multiple knowledge sources, such as textbooks, manuals, WWW, and subject expert.

Knowledge acquisition module (KAM) involves the general process of domain model formation (knowledge that is required to perform a particular task), model instantiation and compilation, elicitation and finally, model refinement (Chandrasekaran, 1987), (Wielinga et al., 1992), (Musen, 1989). This involves representation of both epistemological, inference and problem-solving knowledge about a domain are represented explicitly.

Knowledge acquisition is difficult and time-consuming (Hoffman, 1987). It is therefore difficult to reuse domain knowledge if new system is developed. Although, both manual and software techniques have been developed for acquiring knowledge from expert (Hoffman, 1987), (Boose 1988), (Shaw and Gaines, 1986). The development and acquisition of knowledge as well as its efficient utilisation during instruction may help in achieving the reusability of ITS components.

The Knowledge acquisition module is implemented as software agent (Russell and Norvig, 1995) that interacts and monitors the learners' activities during the instruction. To accomplish this task, the KAM must be interfaced with both the learning environment and the case-based ITS. The details of this process will be discussed in Chapter 4.

3.6.1 Integrating Pedagogical Agents

Pedagogical agents (Lester et al., 1999), (Rickel and Johnson, 1999) are used to provide richer learning and interaction techniques in a learning environment. Pedagogical agents offer an enhanced approach for broadening the bandwidth of tutorial communication and for increasing a learning environment's ability to engage students during instruction (Elliott et al., 1999). Pedagogical agents may offer advantages over conventional intelligent learning environments because they enable closer and more natural interactions between students and the courseware. Such an approach is in contrast with the static text documents that characterise some instructional material.

An agent is "something that perceives and acts" (Russell and Norvig, 1995) and they are "intelligent/autonomous" (Petrie, 1996). Within this context, the notion of agent is quite broad. Because agents are essentially used to solve a problem, they are characterised with different attributes and behaviour and may be used to provide support for lower level activities.

The pedagogical agents paradigm involves the interface agents (André, 1999), (Ball et al., 1997), (Hayes-Roth and Doyle, 1998) and pedagogical agents (Johnson et al. 1998). The interface agents provide an enhanced metaphor for human–computer interaction. The interface agents approach also provides a means for adapting instruction to individual learners and communicating with students through mixed-initiative, tutorial dialogue (Goldstein, 1976), (Burton and Brown, 1982) or as a learning companion (Chan, 1996). Pedagogical agents can also increase the student's motivation to perform a task (Lester et al., 1999), (Walker et al., 1994), thereby providing a tutorial control.

The pedagogical agent can demonstrate how to perform actions (Rickel and Johnson, 1999). It can use locomotion, gaze, and gestures to focus the student's attention (Lester et al., 1999), (Noma and Badler, 1997) and to provide unobtrusive feedback on the student's actions. Furthermore, because pedagogical agents are autonomous agents (able to operate independently from their user) (Castelfranchi, 1995), they inherit many of the same characteristics of autonomous agents. For example, RAP (Firby, 1994), and Steve (Soar Training Expert for Virtual Environments), (Johnson et al., 1998) have been used to create agents that can seamlessly integrate planning and execution, and adapt to changes in their environments. They are able to interact with other agents and collaborate with them to achieve common goals (Müller, 1996), (Tambe, 1997). For pedagogical agents to be effective instructional aids, they must, exhibit robust behaviour in different environments and on different platforms. Also, all the agents activities and events must be co-ordinated in a coherent fashion. However, the need for pedagogical agents to support instruction imposes additional requirements over conventional agents. The pedagogical agent requires a deeper understanding of the rationales and relationships between actions than would be needed to simply perform the task (Clancey, 1983).

Moreover, Herman the Bug (Lester et al., 1999) inhabits a Design-A-Plant learning environment. Herman the Bug provides problem-solving advice to students as students interact with the system. As the student builds the plant, Herman observes their actions and provides explanations and hints. Also, PPP Persona (André et al., 1999) gives on-line help, and guides the learner through Web-based materials, using pointing gestures to draw the student's attention to elements of Web pages, and provides advice via synthesised speech. As the student interacts with the system, the underlying PPP system generates multimedia presentation plans for the agent to present. The agent then executes the plan adaptively and responds to the students' questions.

Research on pedagogical agents is growing fast and spans different domains (Lester et al., 1999), (Rickel and Johnson, 1999), (André et al., 1999). The major advantages for using pedagogical agents in the GeNisa include the following:

i. Pedagogical agents provide multiple levels of advice and combine multiple modalities, which may yield greater improvements in problem solving than less expressive agents (Rickel and Johnson, 1999).
ii. The benefits of animated pedagogical agents increase with problem-solving complexity (Rickel and Johnson, 1999). As students are faced with more complex problems, the positive effects of animated pedagogical agents on problem solving are more pronounced.
iii. They allow explicit planning of tutoring that involves sub-goals, which can be annotated as a solution to the current task and can indicate which plan has been achieved.
iv. They support the student's problem-solving activity by reducing working memory load by providing hierarchical structured task plans
v. They can be used to demonstrate explicit problem-solving methods; for example, animated pedagogical agents can be used to demonstrate how to design a plant (Lester et al., 1999).

Animated pedagogical agents can provide both verbal feedback and non-verbal dialogue to the student and to provide more varied degrees of feedback than earlier tutoring systems (Johnson et al., 1998).

Commercial packages are available for integrating verbal and animated characters into a learning environment. An animated persona (Johnson et al., 1998) was used to provide animated characters with the GeNisa learning environment. The animated character employs the behaviour–space approach for animation and uses speech synthesisers for voice. A behaviour space consists of a library of "behaviour fragments" that are used to generate the agent's behaviour. A behaviour-sequencing engine then dynamically strings these behaviour fragments together at runtime (Johnson et al., 1998).

3.6.2 A Web-Based Instructional Environment

The increasing use of ITS has resulted in a growing demand for ITS applications to be more intuitive and dynamic, and have platform independence (Atolagbe and Hlupic, 1998). Current research in ITS is indicative of the contemporary trends in using hypermedia and the Internet for

educational purposes (for example, (Brusilovsky, 1998), (Brusilovsky et al., 1997), (Weber and Sprecht, 1997), (Towne 1997)).

There has been considerable impetus for ITS researchers to identify methods of ensuring that students are provided with resources for teaching and learning in a collaborative and dynamic environment (Murray, 1999), (Brusilovsky, 1998). Hypermedia systems, such as the WWW, Hypertext Marked Language (HTML), and Hypertext Transfer Protocol (HTTP), (Berners-Lee, 1996), holds great promise as a medium for developing and deploying courseware applications. The WWW approach provides the added advantage of developing and presenting instruction on heterogeneous platforms (Brusilovsky, 1996), (Brusilovsky, 1998).

Web-based ITS are geared towards providing each of the four components of ITS (Weger, 1987) on the WWW. For example, adaptive hypermedia textbook (hyperbook) (Brusilovsky, 1998), (Brusilovsky, 1996), uses semantic networks to describe the domain models and to index the hyperbook with the corresponding domain nodes. Also, the student model has also been implemented by using HTML forms and Common Gateway Interface (CGI) (Nkambou and Gauthier, 1996). This system uses a curriculum component, a planner, and a tutor to present material to the student. Automatically generated tests are used to capture the student's abilities and the results of these tests are used to construct the student model. Furthermore, a method for adaptive presentation of instructional material on WWW has been advocated (Calvi and De Bra, 1997). This approach was used to adapt the page content of the tutorial to the users' knowledge, goals, and tutorial needs.

The WWW adopts architecture of multiple servers and heterogeneous clients for distributing application on different platforms (Brusilovsky et al., 1997). Additional flexibility for the client can be gained through the use of proprietary mechanisms from Netscape called plug-ins, or Internet Explorer scripting language support (Netscape, 2000), (Microsoft, 2000). Furthermore, an Application Programming Interface (API) is supplied, which provides a greater degree of control over the rendering of information upon the client application window. This can be exploited to achieve a customised presentation of instructional material.

This research adapted a client server implementation for web-based ITS (Brusilovsky, 1996), in order to use the server side for application logic and to model the communication processes between the GeNisa components and users. The feasibility of this approach will be investigated

further by using Java programming language, because it provides greater flexibility for application development across platforms (Gosling et al., 2000).

Common Gateway Interface (CGI) and Java make it possible to present the content of an ITS through the WWW (Brusilovsky, 1996). CGI is a standard for interfacing external applications with information servers, such as HTTP or Web servers, for execution of the program (NCSA, 1997). It is possible for a client to send information to a WWW server, whereupon a script is executed that returns an appropriate response to the client. These scripts are typically written using scripting languages, such as JavaScript or Perl.

In general, hypermedia links are implemented through embedded mark-up within WWW documents. This format for WWW documents is called Hypertext Mark-up Language (HTML) (Berners-Lee and Conolly, 1995). HTML provides the syntax for managing document presentation and delineating link anchors together with binding them to a destination document location. The inherent difficulty with using HTML is its general lack of capability for creating complex interactions and dynamic screen displays over the Internet, e.g. simulation, animation, games, click and drag. An alternative approach is to use a Java applet or an ActiveX control to provide hypermedia features in the page. But this implies sending a large number of bytes over the Internet connection. Java on the other hand is more suitable for developing dynamic and static object over the Internet. Java applets are able to access databases directly through a Java-enabled browser and it allows a sessions-oriented connection.

GeNisa uses distributed client/server architecture to enable transmission and delivery of instructional documents over the WWW. The schematic representation of the main components of deploying GeNisa over the WWW is shown in Figure 3.7 (adapted from Atolagbe and Hlupic, 1998). It consists of the following components:

i. *Client Side*. The platform provides an environment in which the users' applications are executed. The student may use a Web browser such as Netscape Navigator or Microsoft Internet Explorer to access the server. This employs client-side applets to provide tightly coupled, fast response interactions. It controls the display of animation and hypermedia objects, and co-ordinates the user's activities. It executes the user's request and communicates with the application database server. It runs the ITS and other application software either as a stand-alone or as a plug-in to Web browsers. It also permits the user to access the same information available in the databases and serves as a means for deploying applications of WWW.

ii. *The Web Server*. The HTTP server accepts requests from browsers through the HTTP protocol (Berners-Lee, 1996). It forwards these requests to other parts of the system to obtain appropriate data and reply to the browsers. It also serves as temporary for hypermedia and data storage. It consists of a Web browser plug-in that allows users to download and run cases from the WWW. The server dynamically creates HTML pages that the user can use, by using common gateway interface (CGI) scripts. The server is also responsible for managing each connected client by maintaining a log of the user's current view, and the particular document displayed to the user. The Web server contains the GeNisa component such as pedagogical expert, student model, and tutorial components. This approach allows GeNisa applications to be executed in either standalone, client-server or on server side.

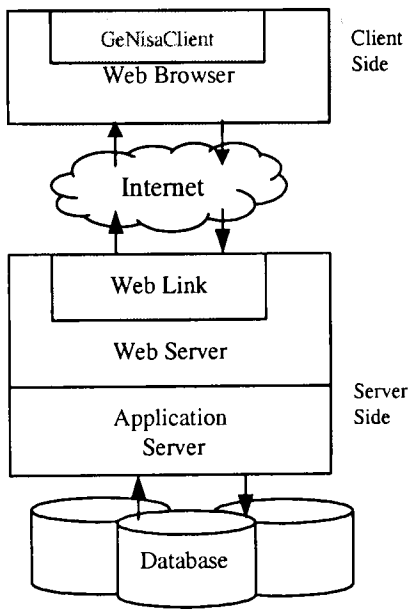


Figure 3.7 Web-Based Learning Environment Components

iii. *Web Link*. It provides a HTML-rendering system that provides access to Java applets from HTML pages on the WWW. It allows clients without the WWW to plug-in and access the system.

iv. *Plug-In*. A Web browser plug-in that allows the user to run an application and access the ITS database through the WWW.

v. *Databases*. The databases contain a repository of case scenarios, assessments and planning databases used during instruction.

Navigation within the courseware is supported by scenario-based navigation, which allows selective presentation and quick access to the HTML documents by following hyperlinks. An HTML page can incorporate Java applets, to be retrieved during instruction. This may enhance

the extendibility of the HTML documents because the link structure can automatically adapt to the database content.

3.7 Summary

This chapter has presented the conceptualisation of the generic architecture for an intelligent training system. The architecture uses a components-based approach to conceptualise each component of the architecture. The generic architecture uses the standard ITS model and components, with the addition of a scenario library and a knowledge-acquisition module. The architecture integrates unobtrusively with the learner's standard learning environment, providing most of the interaction with the users through the user interface.

Issues identified in the previous chapter were examined more closely in light of the objectives for this research and the requirements for the generic architecture. The abstract design of the generic architecture consists of a suite of interrelated abstract classes that embodies an abstract design for each component functionalities. This framework has the advantage of allowing components reuse and it is implementation-neutral.

CHAPTER 4

DEVELOPMENT OF THE GENERIC INTELLIGENT INSTRUCTIONAL SYSTEM (GeNisa)

4.1 Introduction

The previous chapters have presented a design framework for the development of a generic intelligent instructional system architecture. This framework represents a systematic and comprehensive development methodology for ITS (e.g. (Self, 1999), (Murray, 1999), (Halff, 1988), (Clancey, 1992), (Breuker, 1990), (Khuwaja et al., 1994)). This chapter describes the design and implementation of the GeNisa generic architecture.

This research employed component-based approach (Roschelle et al., 1999), (Ritter and Koedinger, 1997), (Roschelle and Kaput, 1996), (Suthers and Jones, 1997) for the design and development of the generic architecture because it may result in simpler and more concise implementations than those possible with previous methodologies. Furthermore, component-based approach supports the object-oriented mechanisms for encapsulation and inheritance thereby offering a great degree of flexibility and reusability (Booch, 1998).

This chapter begins by describing the design of the GeNisa architecture and the transformation of the conceptual model into architectural components. This is followed by description of the GeNisa design and learning environments. Section 4.5 describes a simulation-modelling example, and section 4.6 describes this research approach for deploying ITS over the WWW. Finally, section 4.7 provides a summary of this chapter.

4.2 Design Environment for the Generic Architecture

The GeNisa architectural consist of different elements such as software components, communication mechanisms, states, processes, threads, events, external systems, and source code generator (Garlan and Shaw, 1993), (Kruchten, 1995), (Taylor et al., 1996). Interactivities between these elements are used to address other issues such as message passing, data flow, resource usage, state transitions, and temporal orderings.

The key design consideration are component modularity, flexibility, ability to add component functionality incrementally, and ability to inter-operate with commercial software and Internet resources (Roschelle et al., 1999), (Brusilovsky et al., 1996), (Roschelle and Kaput, 1996), (Murray, 1999), (Ritter and Koedinger, 1997). These design considerations are essential because they address fundamental issues of adaptation and evolution of the software components. Furthermore, the functionality of each components of the architecture has been determined in consistency with the design principles discussed in Chapter 3, on the basis of analysis of reviewed literature (described in Chapter 2) and object-oriented and AI methods. The justification for this approach is to allow components modularity, reusability and to provide design and implementation support for expressing modular designs concisely (Sommerville, 1996). Therefore, an object-oriented software development paradigm (Booch, 1994) has been used to separate components implementation details from the execution environment (Gamma et al., 1995), thereby supporting the integration of multiple heterogeneous component modules into an application. For example, Microsoft Windows uses the Microsoft Foundation Classes (MFC) to implement Graphical User Interface (GUI) components (Bugg, 1999), (Prosise, 1999) and the Java language uses the Abstract Windows Toolkits (AWT) (Gosling and McGilton, 1995) as user interface toolkit. Moreover, each component in the generic architecture has been implemented on the basis of assumptions discussed in previous chapter and its configuration (e.g. algorithm and data structure). The essence is to promote independence of reusable elements between components by: (i) increasing the flexibility to compose those elements; (ii) focussing on modularity of systems and on the reasoning subsystems. The rest of this section present the method developed and used in designing the GeNisa design environment and discusses the main components of the architecture.

4.2.1 Unified Modelling Language

UML (Rational, 1998), (OMG, 1999), (Rumbaugh et al., 1991) has been adopted in this research because it allows the feasibility of the generic architecture to be investigated from a broader perspective. This permit this research to explore the advantages that might be realised for developing ITS.

UML notations consist of three main specifications (Rational, 1998), (OMG, 1999): (i) a notation guide that specifies the visual appearance of UML diagrams, (ii) a semantics specification that details the UML meta-model, and (iii) the OCL (Object Constraint Language) specification that adds a first-order predicate logic language for expressing constraints on UML models (OMG, 1998). However, for UML to be effectively utilised, it needs to be used in conjunction with an

object oriented analysis and design method (Ambler, 1998). This approach can facilitate more concise design representation, and more flexible reasoning for developing ITS. UML allows modelling ITS components by extending UML activity diagrams, in order to model various aspects of typical domain activities before development. In addition, it may provide means for synchronising development activities such as managing user interfaces and forms. This approach provides a rich construct for visually specifying and developing architecture components. Besides, UML provides the following advantages: (i) object orientation can be used to model different components (Booch, 1994); (ii) it provides a generic framework for analysis and developing components, and can also be used for documentation of the early stages of design; (iii) UML is a recognised standard language for object oriented development (OMT, 1998), (Rumbaugh et al., 1991), (Booch et al., 1999a).

The object oriented analysis and modelling approach used in this research involved a combination of used cases (Jacobson et al., 1992), Object modelling (Rumbaugh et al., 1991), Statecharts (Rational, 1997), (Ambler, 1998), and event sequence diagrams (Rumbaugh et al., 1991), (Jacobson et al., 1999). The design notation is based on the UML (Booch et al., 1999b).

As stated in Chapter 3, UML elements and notations have been adapted for use in this research. This approach allows (for example) the functional requirements of the system to be defined in terms of use cases and actors (Rumbaugh et al., 1991); also classes can be defined in terms of their attributes, and their relationship with other classes. Its important to note that UML is only a standard *notation*. It does not describe software development processes (OMT, 1998). The development description, or *method* used in this research consists of five-phase framework outlined in subsequent sections (depicted in Figure 4.1).

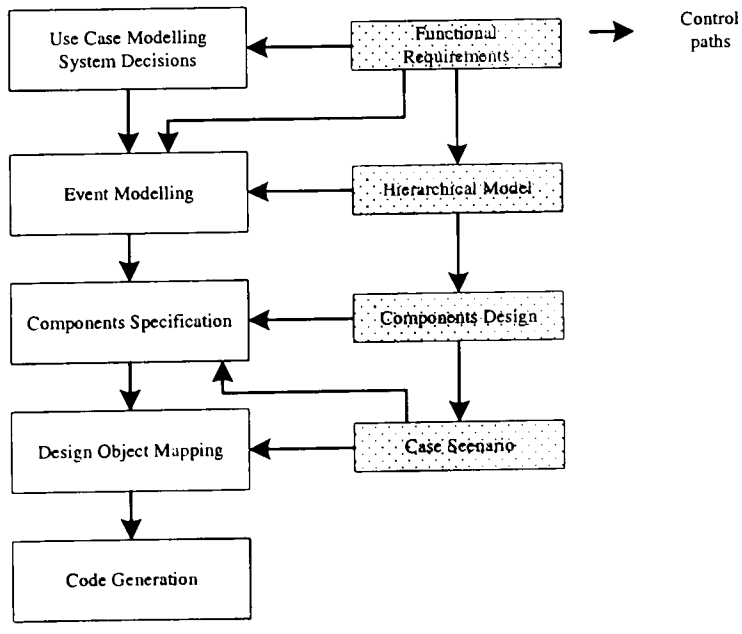


Figure 4.1 Model of GeNisa Design Environment

The functional model of the GeNisa design environment is based on the operational structure of the UML processes (shown in Figure 4.1). This involves object oriented representation of the design environment components by using UML notations through use case diagrams, and sequence diagrams (Booch et al., 1999a). Each phase consists of the functional model, which involves:

i. *Use Case Modelling*. A use case describes the functional requirements of the system (Rational, 1997), (Jacobson et al., 1992), (Ambler, 2000). A use case depicts the interaction between the user (the actor) and the system (shown as a box). An example of use-case diagram is depicted in Figure 4.2. It depicts the *actor*s (person, organisation, or external system). Actors are shown as stick figures and use cases are shown as ovals. In this example, students are logging into the courseware through the tutor. Developer/Tutor inputs tutorial, and is responsible for course administration. Developer/Tutor and student access the course documentation. The functional representation (shown in Figure 4.1) serves as input into this stage. Functional Representation stage involves capturing application goals, in addition to the functional and data related requirements of applications.

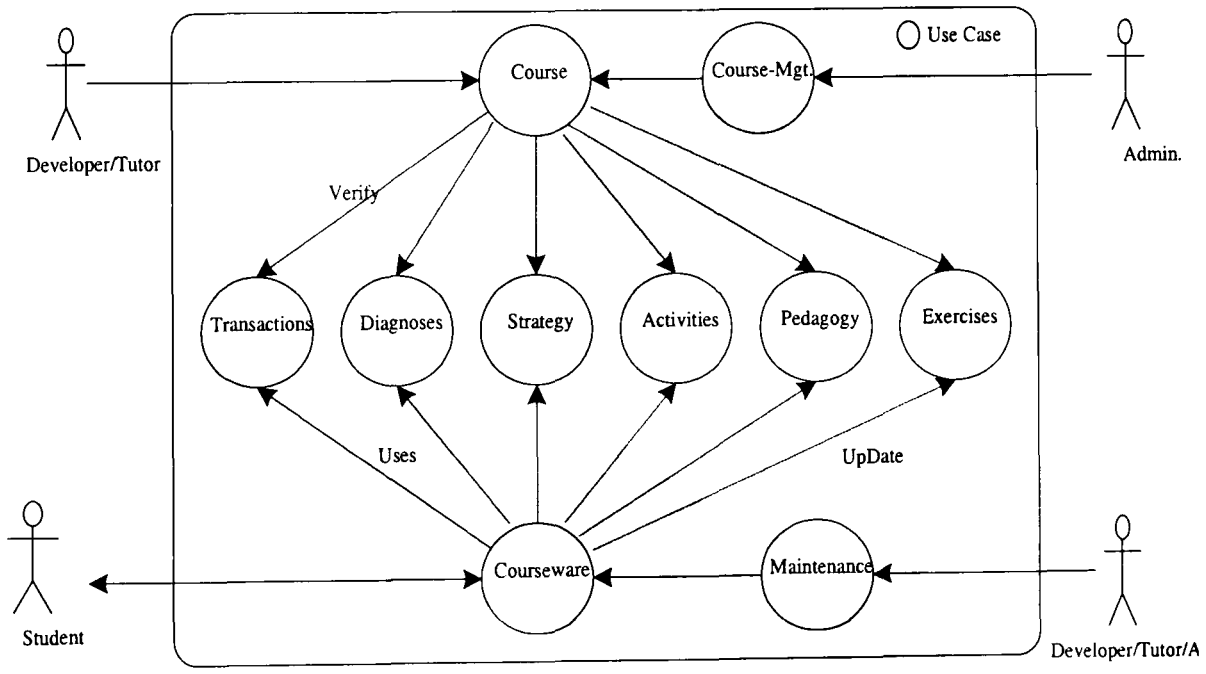


Figure 4.2 Case Diagram for a Courseware

A *use-case scenario* (depicted in Figure 4.2) is a set of scenarios associated with each sub-process (represented by the use case diagram). Scenarios are represented by means of sequence diagrams, which describe the behaviour and interaction with other entities (actors, internal components).

ii. *Event Modelling*. Events required between components and to handle exceptions. It allows events handling between different components to be made explicit and separate from the components' definitions. It involves behavioural specification of static semantics and inter-component consistency constraints. Moreover, the event modelling depends on the structural model of the application. The Structural model (Rumbaugh et al., 1991), (Ambler, 1998), (Booch 1994), defines the classes of the system and their interrelationships (including inheritance, aggregation, and associations). Structural models are used to define what the system will be able to do and how it will be built. It also shows associations between classes. For example, Figure 4.3 depicts the structural model of a computer based tutoring system.

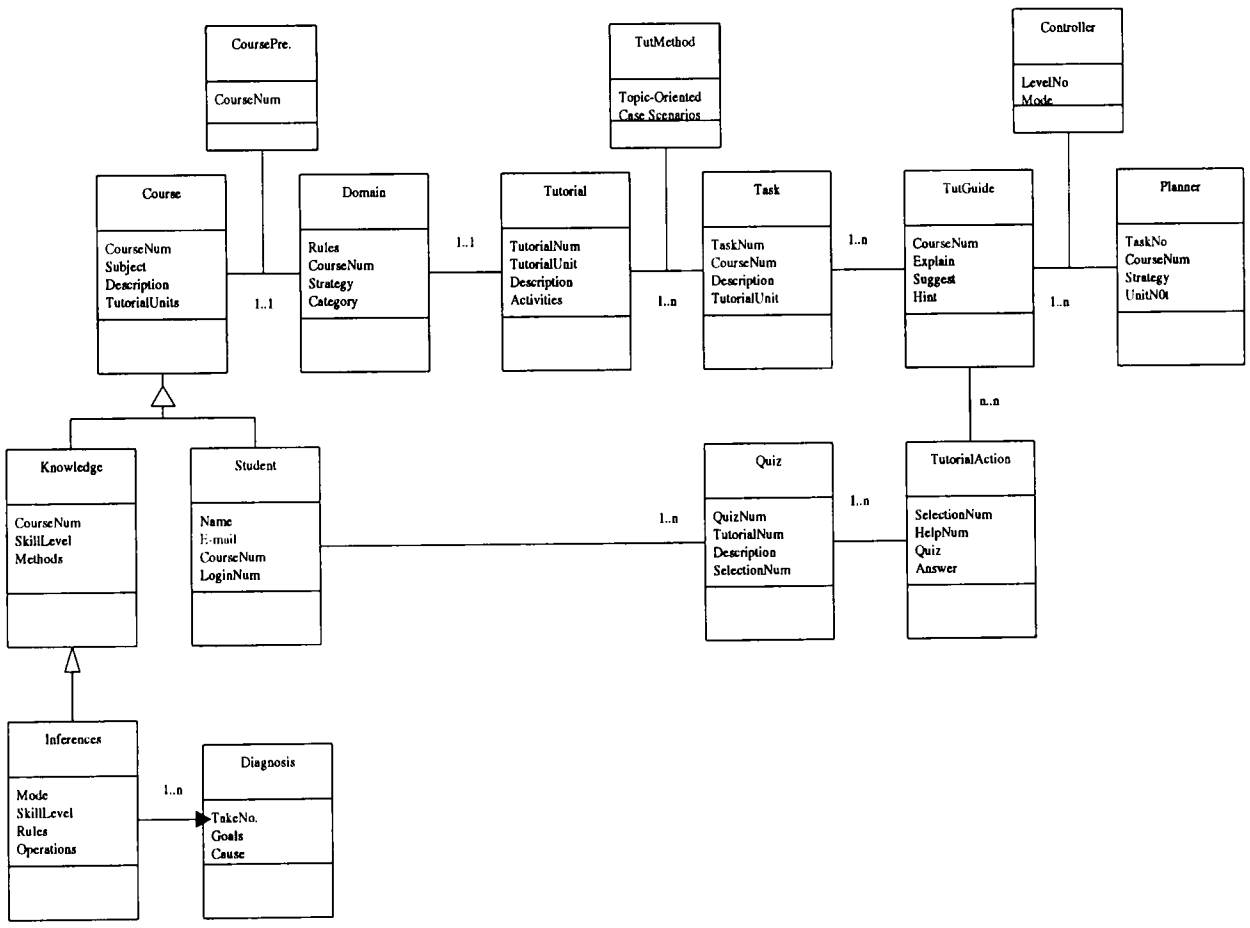


Figure 4.3. Structural Model Diagram of a Tutoring System

iii. *Component Specification*. Component specification and the behaviour are designed so as to allow objects to be mapped into an application model. The different components of the application and relationships among them are identified. At this stage, the developer specifies the detailed ITS components structure and its interfaces to other applications.

iv. *Design Object Mappings*. Represents method for defining component behaviour. They define mappings for input and output between components and the behavioural representation. This involves using *sequence diagrams*. A *Sequence Diagram* (Rational, 1997), (Jacobson et al., 1992) is generally used to represent the logic for a use-case

scenario. The sequence diagrams depicts (Figure 4.4.) the logic behaviour of the system in the context of a use case scenario, which can supports component-based development. Each box represents an object, while a line represents an event communication path.

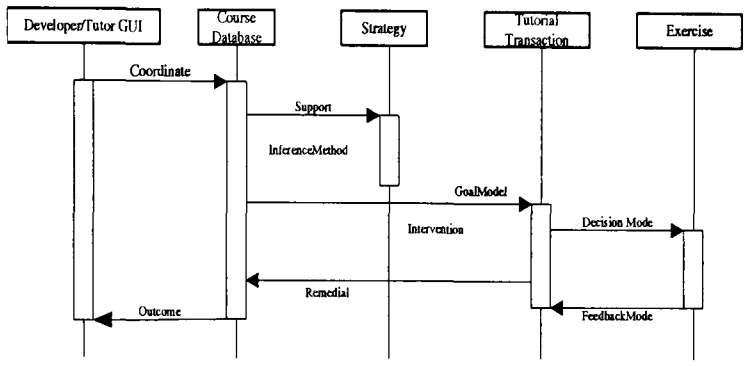


Figure 4.4 A Sequence Diagram for a Tutor

v. *Code Generation.* Code generation in GeNisa is supported with a language-independent abstract base class and Java-specific subclasses. The class Generator is an abstract base class similar to the design pattern (Gamma et al., 1995). Generator class (Java Class) is a Java-specific subclass that generates Java source files. The code generation procedure requires data that are obtained during static semantic analysis. Each entity creates ClassType objects, which generates the bytecodes for each class.

This framework may enables developers to map different entities into application specific components during development (Musen et al., 1995). Furthermore, the rationale behind this architectural design is based on design patterns, which are implementation independent descriptions of component classes (Gamma et al., 1995), (Pree, 1995), (Buschmann et al., 1996). This framework has been adopted to increase the comprehensibility of the program by other developers, and to reuse application classes.

GeNisa is designed to run across platforms, and the mechanisms of the platform requirements have been abstracted into class libraries to enhance GeNisa's portability. The GeNisa interface is implemented within a Java graphic library (Gosling et al., 2000), (Arnold et al., 2000), which allows the generic architecture program to be easily configured on different platforms.

The generic architecture implementations of UML meta-model were initially generated from Rational Rose™ model. This approach allows generic architecture to inherit UML meta-classes and attributes. Also, the UML diagrams use the PGML (Precision Graphics Mark-up Language) standard file format to store DesignObject (W3C, 1998). This provides the added advantage of

improved design representation and UML diagrams may be viewed on different platforms. Furthermore, this framework represent a different approach over previous techniques such as (Mizoguchi et al., 1996), (Van Marcke, 1992), (Murray, 1996). The framework supports the evolutionary nature of the components design process and may satisfy the cognitive needs of the user (Anderson, 1990).

4.2.2 Design Environment Class Library

The GeNisa design environment offers a unified infrastructure for components development. The overall architecture consists of a set of class libraries that provide generic support for components design and a set of packages that provide more specialised support for domain specific computation. Examples of the former include packages that contain math libraries, graph algorithms, and interfaces to Internet components.

All component modules are encapsulated by Java wrapper code so that users of the class library see a normal class structure with data structures and methods that can be extended (Booch, 1994), (Gosling et al., 2000), (Arnold et al., 2000) and are unaware of the underlying low-level implementation. Furthermore, the GeNisa design environment consists of different classes of library that encapsulates a list of commands the user needs to create different components and view component data. All classes implement serialisable interfaces, which can be executed on a target application. This approach allows the developer to extend each component class library. The key architecture class libraries is depicted in Figure 4.5. The key syntactic elements are boxes, which represent classes, the hollow arrow, which indicates generalisation, and other lines, which indicate association. Some lines have a small diamond, which indicates aggregation.

The class library contains a range of classes that corresponds to the entities in the design environment. The library consists of classes for both graphical and textural representation of component objects, assessment, design assistant, and inference engine, with a carefully defined interface. From this library, one or several component methods can be selected in an application, and used to drive the behaviour of a component. The Actor interface extends the capability for transporting data through their connectors. The interface can initialise the execution of component's methods. Other essential classes include:

i. *DesignObject.* Represents ports, nodes, and edges. Ports are connection points on nodes, and edges go from a source port to a destination port. Ports allow the developer to represent the semantics of the DesignObjects types. DesignObjects (shown in Figure 4.5)

are connected together by ports and can communicate with each other by sending and receiving event objects through these ports. DesignObject from one project can be imported into another project. The import is done by referencing the design objects only; this allows the classes to be reused.

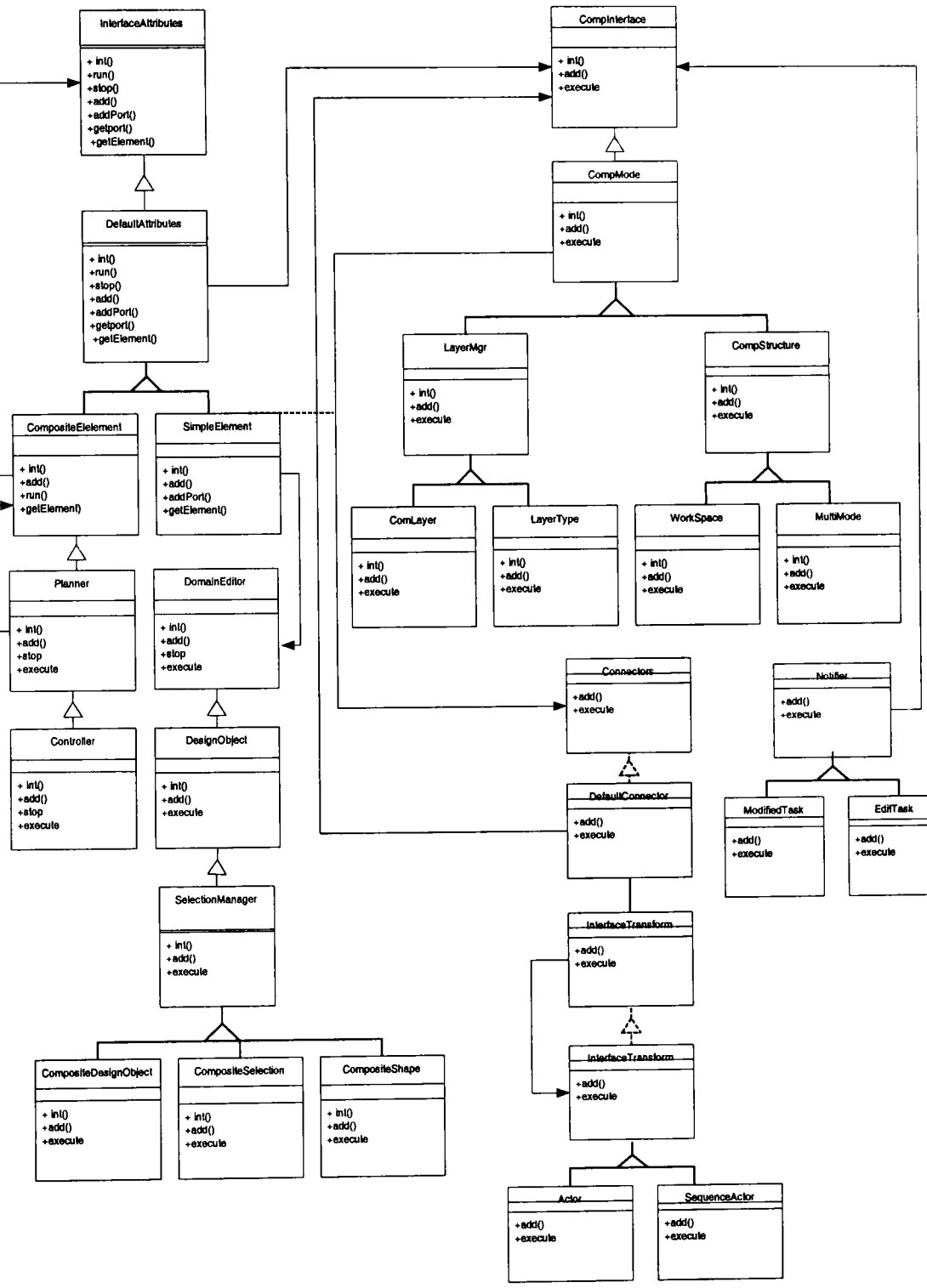


Figure 4.5 Section of GeNisa Design Environment Class Library

ii. *CompositeInterface*. Maintains consistency across hierarchical objects and their relationships. An automatic change propagation facility allows developers to use a new version of class module with an existing design.

iii. *CompositeStructure*. Holds the other classes together and routes event messages among them. It permits definition and hierarchical representation of design objects. It can also be

used to invoke other tools and for displaying tools diagram. Other view functionality is provided by Modes and Selections classes, which also provide graphics for interaction feedback. The Modes and Selections class libraries primarily play the role of controller; and perform some event-handling operations.

iv. *CompModel.* Manages the mapping from DesignObjects (depicted in Figure 4.5) in a diagram to application objects. GraphModels interpret existing data structures as graphs. This approach is similar to Java's Swing user interface library for implementing tree widgets and tables.

v. *SelectionsManager.* This keeps track of which DesignObjects are selected and the effect of the event process; for example, SelectionResize allows the bounding box of a design object to be resized.

vi. *CompositeElement.* This represents objects that process user input events (e.g. mouse movement and clicks), and executes event commands to modify the DesignObjects, e.g., dragging in ModeSelect shows a selection rectangle.

vii. *Layer.* This encapsulates the DesignObject, modifies and can extend object behaviour and functionality.

Each class library consists of component module that have identity, a method and attributes (Booch, 1994), (Sommerville, 1996). Also, each component encapsulates their methods and behaviour, and interacts with each other by calling each other's method. Interactions between components are defined through their interfaces i.e. through the names of method and their parameters.

Whenever a component in the ComEditor changes in any way, it calls its Notify() method, which sends the Update() message to each of the CompMap components which are observing it. In this way, the visual image of every DesignObject component is always easily and efficiently updated to reflect state changes, even if there are multiple views (CompMap graphs) or multiple occurrences of the image of a Constraint Graphs component on a single view.

4.2.3 Components Events Manager

The events manager maintains knowledge about the software process, and the particular of state a component development and instructional activities. In doing so, it can guide developers through essential tasks that are necessary, automate particular tasks and most importantly, control the way multiple components interact. Figure 4.6 depicts the generic architecture events communication manager. The User Interface imports/exports a number of component-specific window classes,

based on event context in response to event received from the user interface management system (UIMS). The ToolKernel encapsulates basic functionalities such as creating new application, opening a file, deleting a file, keeping track of the currently opened editors, etc. The ControlModule controls a component execution and translates various incoming events, such as user input, into calls of operations executed by underlying components.

The key to maintaining coherent behaviour during user activities is to maintain a rich representation of context for each component. Therefore, each component event is designed to run in as separate module, communicating with different components by using an inter-process communication links.

The Tool Kernel (shown in Figure 4.6) is a library of classes and subsystems that encapsulate the basic component functionality. One of the purposes of the Tool Kernel is for managing the set of components that are displayed by the system. The Tool Kernel subsystem implements the execution of design assistant, control mechanisms, "to do" list, wizards, and the user model and offer operations to the component class that are used for reacting to user-input as well as to service requests.

The Tool Kernel subsystem has been implemented so that it can be reused among arbitrary tools. During execution, each component creates an object of design class and stores it in an instance variable of the Control module. To invoke events exported by the Tool Kernel, the Control class invokes the method from this object.

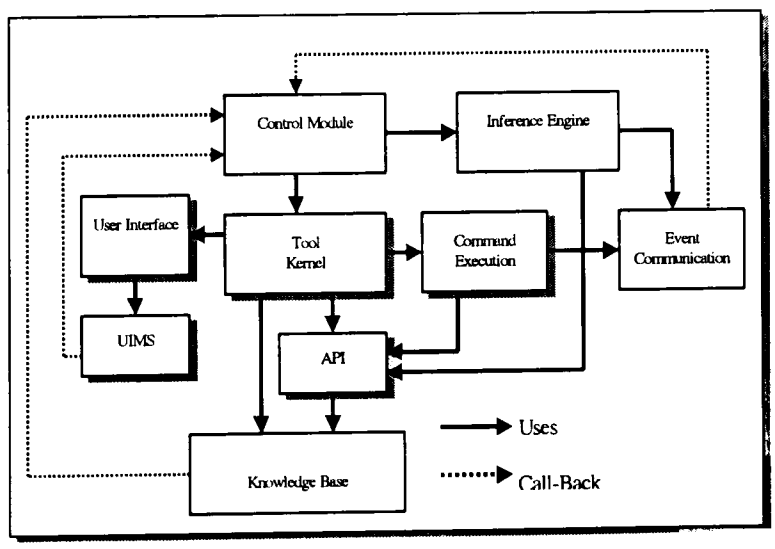


Figure 4.6 Events Communication Subsystem

4.2.4 Design Components

Part of the requirement for the generic architecture is that there must be a clear separation between the component development and instructional environment, and their underlying events. This requirement addresses the flexibility and generality of the generic architecture by allowing utilisation of different components classes and configurations with differing packages, through the user interface.

In an attempt to address the issues associated with providing dynamic and greater flexibility in a platform-neutral environment, Java programming language (Gosling et al., 2000) is used to implement the user interface. This approach allows Java applets to run on heterogeneous platforms. This approach allows the user interface to be customised as required by the user. For example, a tool developer may only be interested in using the design tool in the development environment and not in the courseware.

The class library provides Windowing and interface event in which the components class library objects interact across platforms (Booch, 1994). It also provides high level abstractions, including modal dialog boxes and non-modal frames, inter-application communication protocols, and the wide range of data structures required by the different applications. This approach exemplifies the possibility of decomposing common modules from many different applications and reusing them in distinct implementations. The generic architecture has been designed as a Windows-based component development/learning environment. A Windows-based user interface was chosen to minimise "cognitive load" of learning the system and to keep information visible and easily accessible to different levels of users (Shneiderman, 1998), (Nelson, 1996).

The design and implementation of the generic architecture user interface are guided by the usability guidelines from (Shneiderman, 1998), (Constantine and Lockwood, 1999), (Gosling et al., 2000), (Arnold et al., 2000), (Apple 1993). The top-level functionalities of the generic architecture GUI are:

i. *Document Management* supporting the users' concept of a single document file as the basic unit that is edited and communicated to other users.
ii. *Editor Support* through a simple and natural user interface, as well as maintaining the content of the document and its annotation, also keeps track of what changes of contents for versioning purposes.

iii. *Filing Facilities* that, in addition to storing the document and annotation, also store tables of version information as separate resources in the same file as the document.

iv. *Interchange Facilities* that allow the import and export of components in Graphical Interchange Format (GIF) allowing users to move files between the generic architecture and commercial applications.

The user interface is organised around its main segments: the Main Window, Explorer, Editor, Critique, Tasks List, Design Editors, Property Sheets, and the Output Window. This default set of windows, Workspaces and tools can be adjusted freely to match the user's preferences. The Main Window is the first item opened when GeNisa is launched. The Main Window can be viewed as the "control centre" of the design environment. All important operations and actions are accessible from this window. The Main Window can be broken into four separate groups of controls: the menus, the toolbar, the Component Palette, and the Workspace tabs as depicted in Figure 4.7.

Authoring process consists of the following: (i) choosing the "New" action either on the toolbar or from the File or "New" menu. A window will appear for selecting the template from which the user can create the new object, and (ii) in the dialog that appears, the user can enter the name of the new object and select the folder/package in which the object should be created. Template-like structures have been used to abstract commonality between component classes. Each of the Java-specific classes implements methods that generate source code for design elements of a given type automatically and contains (i) the block of variable declarations for the components on the form, (ii) the method InitComponents(), in which all the form initialisation is performed, and (iii) header (and trailing closing bracket) of all event handlers.

The source code is stored in either in a Java file, which contains the (partly) generated Java source; or an XML file, which stores the properties and layout constraints of JavaBean components on the form. The developer can edit the Java files using external editors.

The GeNisa data model is similar to that of most ITS authoring systems (Murray, 1999). But, one particular distinguishing feature is that multiple views upon a single object can be created, allowing different aspects of the object to be focused on. This additional level of abstraction is achieved through the introduction of views and by using UML.

The generic architecture user interface consists of graphical development environment, which provides a supportive environment for application development. The GUI is structured so that all DesignObjects are edited on the upper-right pane. The upper-right pane is also used for displaying the table of current design. The rest of the GUI is used for displaying and manipulating design properties, a "to do" list, view source code, etc., as shown in Figure 4.7. Other essential design components include:

i. *Wizards*. The class Wizard is an abstract base class for non-modal wizards. The wizard control implements a framework for wizard-like dialogs. The framework handles the back/next/finish buttons and the display of the wizard control's pages. Each page of the control can be enabled or disabled. The pages of the control may be edited in a way similar to the tab control.

ii. *Design Navigation*. The design navigation and view tools implement the Model–View–Controller design pattern (Krasner and Pope, 1988), (Gamma et al, 1995) where the view and controller roles are represented on the user interface widget, and the role of the model is represented by a planner class. It provides task-specific views of the underlying design representation and design properties. The planner classes observe the design representation and react to change notifications by sending their own change notifications that cause the view to be redrawn (Eckstein et al., 1998). Navigation tools consist of other user interface objects such as a tree widget, table views, etc.

Figure 4.7 Screenshot of GeNisa Development Environment

iii. *Goal-Directed Search.* Class FindDialog defines the layout of the widgets in the search window. Each widget in the top part of the window adds a predicate object that is used to select search results. For example, the name field contributes a PredicateStringMatch object that selects only model elements with names that match the pattern entered in the name field. The individual predicates are combined into a PredicateFind object that performs a logical-and to select only those model elements that satisfy all predicates.

iv. *Components Design Assistant.* Design Assistant components are simple agents that continuously execute in a background thread of control (Taylor et al, 1996) are. They analyse the design as the developer is working and suggest possible improvements. These suggestions range from indications of syntax errors, reminders to return to parts of the design that need finishing, and style guidelines, to the advice of expert designers. Design Assistant components are controlled so that their suggestions are relevant and timely to the current user's design task, based on information in user model. Improvement suggestions are placed in the "task list".

Design Assistant provides knowledge support to developers during design processes and provide suggestions on alternatives to design decision. Design Assistant have been used in many domains, such as building architecture (Chun and Lai, 1997), (Fu, 1997), user interfaces (Bonnardel and Sumner, 1996) and medical diagnostics (Gertner and Webber, 1998). Design Assistant may guide a novice user through the development of a complex application. Figure 4.8 depicts a design critic's description of the user's design activities that may require further attention. The critics advise the developer of potential errors or areas that require improvement in their design activities. The feedback addresses issues that the developer may have overlooked.

Figure 4.8 Design Assistant

Table 4.1 summarises the advantages of GeNisa framework. Implicit in this framework is a fundamental epistemological shift: incorporating different component functionalities independently of other subsystems into an interactive environment. This can allow developers to share data, which can facilitate component development.

Table 4.1 A Summary of Advantages of the GeNisa Design Environment

COMPONENTS	Advantages
UML based component design environment.	Ensures that component development reflect necessary level of detail and functionality.
The design framework consists of varieties of components (e.g. case scenario, design object Mappings, events modelling).	Allows flexibility to use variety of representational methods during design.
Each component provides unique set of functionalities.	Provides flexibility during design and allows developers to use different type of components and modify them to meet their needs.
Object oriented based components classes.	Provides extensibility, reusability, and comprehensibility.
Flexible design environment.	The framework provides a more flexible design method, which may result in reduction of the development time.

4.3 GeNisa Learning Environment

This section describes the GeNisa Learning environment and knowledge support components that have been designed as domain independent tools, with the goal of making them both extendable and reusable for heterogeneous applications. This section describes different types of knowledge components and the context in which these tools can be used.

The main functions of GeNisa learning environments are to provide explanation, tutoring, and diagnosis (Goodyear, 1991). GeNisa learning environments provide mechanism for building and utilising the different agents or components within the learning environment (Baker et al., 1994), (Baker, 1992). This framework promotes the acquisition of problem solving skills (Goodyear, 1991) and requires the student to take the learning initiatives (Freedman, 1997) and control how knowledge is presented during instruction. This would probably help them to build coherent models of domain from the subject matter content, which would not only help to enhance their understanding. It would also help to develop their problem solving skills, provide enhanced user's interactivities, help students whenever the need arises (Dillenbourg and Self, 1992) and could greatly increase human-computer interaction (Horvitz, 1999), (Murray, 1996), (Soloway et al., 1991).

However, a leaning environment needs to have all the components of an ITS for enabling the system to offer intelligent help to students and to carry out pedagogic activities. A leaning environment supports procedural and declarative knowledge representation by either using detailed cognitive models built from production rules (VanLehn, 1999), (VanLehn and Jones, 1993), (Anderson et al., 1995), (Anderson, 1990), (Sacerdoti, 1977). Procedural and declarative knowledge representation involves performing the domain tutorial tasks by directly executing the rules (Anderson, 1990). The tutorial tasks are therefore represented as a network of domain tasks (plans). Each plan consists of a set of steps, each of which is either a prerequisite action (e.g. conducting an interview with the hypothetical client before writing the requirement) or the main plan (e.g. model development). Also, there may be ordering constraints among the steps, represented as a set of casual links (Horvitz, 1999), (Horvitz et al., 1998). Each casual link specifies that each step in the plan should achieve a goal, which is a precondition for another step in the plan.

GeNisa learning environment uses case-based ITS approach to support learners during instruction. This approach can help with the acquisition of procedural knowledge (VanLehn, 1996a). For example, ELM-PE (Burrow and Weber, 1996) and ELM-ART (Weber and Specht, 1997) allow the student to access relevant examples during LISP programming exercise and provide explanations of the examples. SHERLOCK (Gott et al., 1996), has been used to teach expert solutions for troubleshooting problems, and provide students with a means for comparing their solutions. These approaches may limit the learner problem exploration because the learner focuses on acquiring the domain knowledge needed to perform the task. Also, having control over the navigational mechanism may allow the leaner to capitalise on existing knowledge. In GeNisa, problem-solving procedures are supported by content specific explanations. This has been implemented by using the student's prior knowledge on the subject matter in order to generate appropriate explanations. Figure 4.9 depicts a schematic representation of GeNisa learning environment. It consists of the following main components:

cspgtaa

Brunel University

Department of Information

Systems and Computing

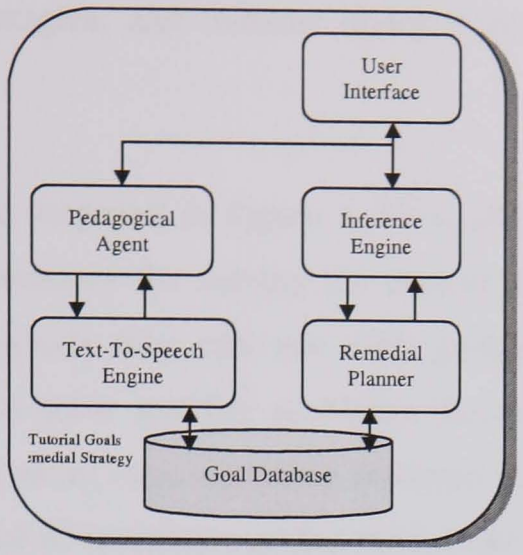


Figure 4.9 GeNisa Learning Environment

4.3.1 User Interface

The user interface consists of dialog handlers, widgets, which are responsible for communicating between the pedagogical agent, discourse module and the user. The user interface displays messages sent to the screen from the remediation planner and it captures student inputs and sends them to the assessment database. The user interface consists of adaptive navigation mechanisms, which can support different needs of the learner (Dieterich et al., 1993). The user interface has been designed to provide flexible and accessible components to the student during instruction. This provides transparent ways for navigation and for interacting with the systems components, which is essential for eliciting knowledge. For example, during instruction, the learner is presented with a description of the case scenario and can navigate around the system and use appropriate components during problem solving. The knowledge support components have been implemented in Java (Gosling et al., 2000) in order to provide dynamic and greater flexibility across platforms.

4.3.2 Remedial Planner

The remedial planner is responsible for deciding appropriate actions to be performed in order to achieve a pedagogical task. It consists of components that elicit the domain knowledge, which is stored in the case scenarios, assessment and planner files. The databases are represented in text/HTML format to allow for editing, portability and reusability. The databases may be easily customised for a particular case-scenario. Remediation planner makes tutoring decisions by examining its own rules and by consulting with the student model. It selects domain topics,

determines tutoring strategies, and initiates dialogue by sending messages to the animated persona.

The remediation module (depicted in Figure 4.10) is also used for presentation of the tutorial knowledge, which is necessary for solving the current case. This approach offers students a structured method by which they can use their problem-solving skill (e.g. knowledge of simulation modelling) to solve real-life problems. Related task-specific information, such as background information about case, operation platform, and user environment are incorporated into the module. The goal of instruction is that the learner ultimately integrates their view of the domain into a correct, coherent, and desired model of the domain (Ohlsson, 1991).

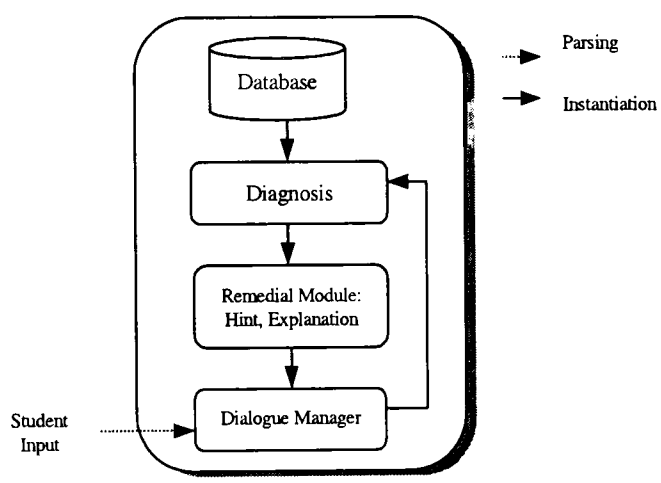


Figure 4.10 Tutorial Remediation

If the user is unsuccessful in achieving the tutorial goals, GeNisa tries to remediate the current solution of the student. On the other hand, if the diagnostic components fail, a default remediation strategy is selected. The GeNisa tutorial remediation (depicted in Figure 4.10) consists of three subsystems: (i) database consists of current beliefs and domain problem scenario, to support current tutorial, (ii) diagnostic module, for providing appropriate intervention to the learner misconception (Siemer and Angelides, 1998), (iii) dialogue manager use for translating student's dialogue into appropriate remedial action: and for providing hints and explanations to the student (Siemer and Angelides, 1998), (Canfield et al., 1992), (Burton, 1988). The dialogue manager uses the problem solving methods (domain database) to guide the learner learning processes. If the diagnostics cycle fails, a default remediation is automatically invoked.

The tutorial "space" has a tutorial goal structure that constitutes a portion of the current knowledge. The tutorial goal structure consists of a set of domain topics, connected to each other via generic didactic links (Wenger, 1987). This approach allows topics to be quickly retrieved and allows for a better diagnosis of the student learning activities. Didactic components are

represented as pedagogy decision-making tools and they are used for selecting tutorial tasks and for remedial interventions. Didactic links are represented as casual relations (Bayesian network) and may be used to set different users' learning goals (VanLehn, 1996b). To be able to respond flexibly to the student, the remediation planner refines its plans according to the student's level and learning activities.

4.3.3 Text-to-Speech Engine

Adding speech-enabled interfaces to the learning environment may enhance learner's interactivities by minimising keyboard use. Furthermore, speech-enabled learning environment may increase the usability of software to novice users who have minimum keyboard experience. The system uses Microsoft Text-to-speech engine or synthesisers, to perform speech synthesis by conversion of text and generating spoken language (Microsoft, 2000). Text-to-speech was considered as an alternative to using a digital audio recording, because the latter may be too expensive to record.

4.3.4 Pedagogical Agents

Pedagogical agents provide pedagogical functions: presentation, student monitoring and feedback-probing questions, hints, and explanations (Johnson et al., 1998), (Lester et al., 1999), (Elliott et al., 1999). These capabilities are coupled with an animated persona that supports continuous multi-modal interaction with a student (Rickel and Johnson, 1999). The pedagogical agents guide the user through the case scenario, and provide the following functionalities: (i) instructional support by monitoring student activities and offering hints or an explanation, (ii) an enhanced learner control by allowing the learner to decide when to get an explanation or hint, and (iii) hierarchical presentation of a plan and provision of advice when required.

The pedagogical agent communicates with all the components in the learning environment. It also executes the Text-to-Speech engine, which converts the input text (domain scenario) into speech. The agent provides both visual and auditory input into the learning environment (Rickel and Johnson, 1999), it makes the structure of the domain visible, accessible and it also helps to lead the student through their problem-solving actions and it also tracks students' responses to quizzes. The pedagogical agent may also employ multiple media in order to improve interactivity with the learner. It also tracked the students' actions throughout the problem solving activities.

4.3.5 Inference Engine

The inference engine communicates directly with discourse and reasoning subsystem and it performs all monitoring and decision making. Inference engine also communicates directly with the student model, a case scenario, to perform the requested operation by invoking the appropriate inference tool.

Learner's input or query during instruction is first analysed by the user-input analyser, which extracts the inference information and the knowledge source. This information is then directed to the inference engine, which, in response, may invoke appropriate content knowledge. This approach provides rich information about the student's cognitive state during problem-solving, and can more readily provide information with which to build a student model or determine the pattern of response selection.

4.4 Knowledge Elicitation and Tutorial Management

Knowledge elicitation and tutorial management is represented in the pedagogy expert. The pedagogy expert provides consistency, coherence, and continuity to the learner during instruction (VanLehn and Jones, 1993), (VanLehn, 1988), (Clancey, 1987), (Wenger, 1987). (Essentially, the pedagogy expert contains a set of specifications of what instructional material the system should present and tutorial management). In GeNisa, the pedagogy expert provides assistance to the student, monitors and criticises the student, and selects problems and remedial material for the student (VanLehn, 1996a), (Halff, 1988), (Clancey, 1987), (Wenger, 1987).

GeNisa uses interactive dialogue between the learner and the system to elicit domain knowledge from the pedagogy expert. The pedagogy expert contains a specification of the curriculum knowledge (Lesgold et al., 1989), stored independently from the tutor, and loaded during instruction. However, the domain expert alone may be inadequate to perform pedagogical tasks effectively (Lesgold et al., 1989), (Anderson, 1988), (Breuker, 1988), because it lacks the knowledge required for pedagogy expertise (Wenger, 1988). Therefore, the GeNisa learning environment uses both domain and pedagogy expert (Murray, 1999), (Wenger, 1988) to support instruction.

The nature of the learning activities that the student is required to perform is goal-oriented (Rubtsov, 1993), (Davydov, 1988), which may help the learner to develop meta-cognitive skills

(Margolis, 1993). Furthermore, goal-oriented learning may increase the students' problem-solving skills, because cognitive skill can be acquired by performing the actual tasks (VanLehn. 1996a).

Wasson (1996) proposed planning of tutorial content and delivery based on explicit representation of the target knowledge concept structure, i.e. curriculum to be performed at different levels of granularity. GeNisa uses domain independent heuristics to structure the presentation of particular concepts. Examples of domain-independent heuristics are: (i) solving basic problem before complex problems, (ii) annotating each problem-solving task with an appropriate component, and (iii) proving appropriate and meaningful hints or help messages.

Task complexity can be measured by the number and entry of preconditions in the heuristic rule. Additionally, simpler concepts may be proper subsets of more complex concepts; this represents a prerequisite for the more complex concept. The progression from simpler to more complex concepts defines a hierarchical structuring of the overall concept space. It is important to note that the curriculum extraction process develops a baseline curriculum; the actual curriculum will be customised during the training session by the student model and the actions of the student.

4.4.1 Pedagogical Technique

Pedagogical technique provides scaffolding of domain knowledge in order to accommodate for the learner' problem solving methods and for self-explanation (Conati et al., 1997). The representation of the knowledge base and pedagogical tasks as "plans" allows the pedagogical techniques to be provided for individual learners during instruction. Essentially, it consists of didactic processes, which incorporates adaptive instructional strategies. This permits the provision of pedagogical intervention to support the learner activities, which can facilitate the development of skills and knowledge of the domain task.

The generation of pedagogical explanations elicited by the system is based on: (i) transformation of the pedagogical knowledge (structured knowledge of the domain tasks) into functional knowledge and components (Atolagbe and Hlupic, 1996), and (ii) by using knowledge of the students' problem solving methods. This approach allows students to make their own planning decisions, and self-explain the pedagogical tasks (Chi et al., 1994).

In order to provide appropriate pedagogical intervention, GeNisa uses the information obtained from the student model with reference to the followings: (i) the pedagogical model used for the

tutorial (in order to guide the learner on how to accomplish the tasks), (ii) the current student problem solving method, and (iii) the dialogue mode, (where the tutor allows the learner either confirm or to explain his/her tutorial activities), which is used to provide appropriate remediation. This information is used to support the learner's problem solving methods during instruction and to adjust the parameters for the appropriate pedagogical intervention. Furthermore, this approach allows the student to explore the "problem space" by applying rules in the student model until the learner encounters a problem. This form of pedagogical interventions requires the learner to articulate his/her reasoning, which is used for monitoring and for generating explanation.

The instantiation of the pedagogical task structure is accomplished through the use of domain-specific knowledge about concept complexity and relationships. Essentially, the system targets the student's misconceptions as the students analyse problem scenario and plan their solutions. The pedagogic model has been designed to encourage students to follow their own problem-solving strategies. Therefore, instead of teaching theoretical aspects of the subject matter, the learner can exploit the analogies found in other domains during problem solving. An illustration of this approach is described in the next section.

Implicit in the pedagogical model depicted in Figure 4.11 is a fundamental epistemological consideration, which relates to domain tasks processes and cognitive skill formation. This can help to engineer a pedagogical task focus instruction by relating models to conceptual knowledge formation and tutorial activity. Furthermore, it serves to organise the knowledge so as to account for different levels of tutorial outcomes and remediation. Therefore pedagogical activities should support and accommodate differences in the ways students construct their knowledge and should facilitate creative problem solving (Atolagbe et al., 1997).

Pedagogical intervention (implicit within the GeNisa learning environment), can guide the learner problem solving activities and allow the student to self-explain the pedagogical tasks (Conati et al., 1997). The number of intervention provided by the system varies with the users' problem solving tasks. The pedagogy module contains multiple strategies and selects appropriate strategy based on the current problem solving methods and the progress of the student. This approach can improve the learner's performance during problem solving (Anderson, 1990), (Ohlsson, 1993).

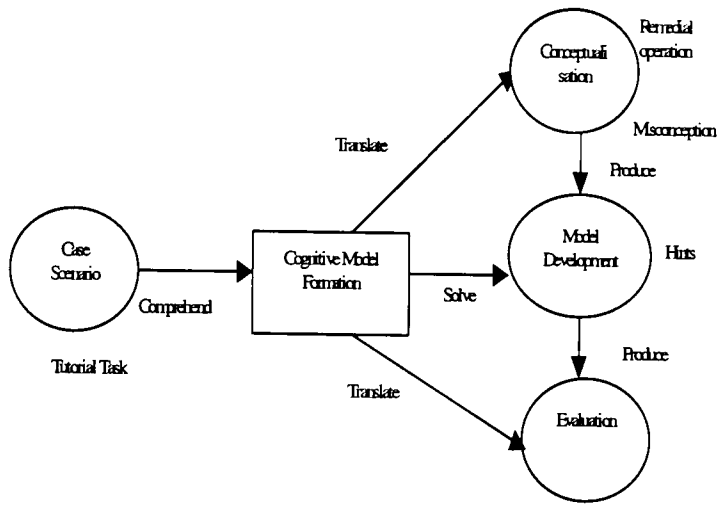


Figure 4.11 Pedagogical Model

The following pedagogical strategies were implemented in order to facilitate the acquisition of domain knowledge. It includes:

i. *Learning with Scenarios.* This strategy uses a real-world scenario as the vehicle for instruction. Presentation of a scenario involves demonstrating the operational activities and teaching the correct methods required to solve the problem.

ii. *Learning by Doing.* Within the context of the scenario, it coaches the student step-by-step operation required to perform the task.

iii. *Practising with Content Feedback.* The student performs activities without prompting by the tutor. When the tutor detects an error or misconception, it provides immediate remediation of the problem.

iv. *Free Exploration.* The user can navigate around a case scenario, without intervention for the system. The learner controls the learning activities.

This approach provides a flexible way of organising tutorial activities, which can improve the students' interactivities. Also, the learner is assumed to initiate problem-solving activities during instruction. This implies that the learner has to generate appropriate questions, answers and explaining the case scenario. This can facilitate the development of higher-level cognitive skills (VanLenh et al., 1992), (Scardamalia and Bereiter 1993), (Scardamalia and Bereiter, 1996). Furthermore, the learner can be considered as having a set of cognitive processes, which are used during problem solving and for generating self-explanations. Hence, the learners would vary the effort they expend on a cognitive process in accordance with their motivations (Chi, 2000), (Chi et al., 1994) and complexity of the problem. By using a pedagogical structure that is guided by

cognitive theory within an interactive learning environment, this can permit simulation modelling knowledge to be developed and to be shared in a collaborative environment.

The tutorial task is represented as a probabilistic model of the domain (VanLehn, 1996a), that is the current domain task, an initial state distribution, a set of available actions, and a utility function "sequence of state". The tutorial task is represented as a set of attributes (variables) with associated probability distributions. This is based on the assumption that a tutorial task consists of sequences of actions. A tutorial action takes place under certain conditions with a given probability and this can influence the type of tutorial content to elicit.

4.4.2 Student Model Components

The control behaviour of the student model is encapsulated as a pedagogical decision-making mechanism. This serves as diagnostic tool for identifying learning problem, and for acknowledging learners' input during instruction. The student model components can adaptively control the tutorial based on the learner problem solving methods (VanLehn, 1996a), which may promote more effective instructional interactions between the student and an ITS. The student model components include:

i. *Character Profile*. Refers to level of tutorial intervention and performance profile of the learner. It relates to the sequence of tutorial actions and assessment of users problem solving activities. The scaffolding actions are determined by the learners' interaction with the components of the navigational mechanisms. The character profile is used to focus the learners' tutorial activities towards most relevant aspects of their ability and can be used to predict the effects of their actions.
ii. *Diagnosis*. It is used for diagnosing student's misconceptions about the subject matter, and providing feedback as required. It encapsulates mechanisms for reviewing the learner's actions and the elicitation of content specific information. It is responsible for interpreting the external data through users mouse actions and generating appropriate responses. It selects an appropriate action based on a repertoire of pedagogical actions such as Show, Hint and Explanation.
iii. *Learning Goals*. A utility functions that guides tutorial actions in order to attain fixed tutorial objectives, but flexible to allow other tutorial interventions. Essentially, the implicit goal is to help the student complete all tasks. Also, the background knowledge (abstract preconditions represented in the knowledge base) is used to select learning biases automatically.

iv. *Management of Tutorial Tasks.* It involves a global representation of the learner in terms of their character profile, diagnosis and learning goals. This approach allows the learner to modify the pedagogical task structure during problem solving, by their sequence of actions and generating appropriate questions and answering it by themselves. The user's actions are recorded through the mouse and the keyboard. Allowing the learner to enhance the pedagogical task has been incorporated into the student model by using a representation of problem solving strategies (represented in the case scenario) and domain specific concepts. This approach allows the learner to adapt their problem-solving methods at different levels of instruction and to modify their decisions in relation to the current problem.

The student model uses probabilistic reasoning to maintain a model of the student's level of competence, and the student's preferred methods of problem solving and learning (VanLehn, 1996a), (Collins et al., 1996), (Martin and VanLehn, 1995), (Conati and VanLehn, 1996). This approach allows GeNisa to use the information in the probabilistic student model to guide a real-time tutorial dialogue and to handle uncertainty in the student model (Pearl, 1988a), (Martin and VanLehn, 1995).

The Bayesian network model provides a probabilistic method for handling the uncertainties in student reasoning (Carbonaro et al., 1995), (Kambouri et al., 1995). It allows the degree of complexity of the domain tasks to be varied, thereby allowing content-sensitive information to be incorporated into the task (Heckerman, 1997).

The structure of the Bayesian network used in this research is depicted in Figure 4.12. The network parameters are derived from probabilities describing the student's domain knowledge, pedagogy task, case scenario, and current problem solving method. These probabilities provide priors for rule nodes and parameters that automatically define the conditional probabilities in the network (Conati et al., 1997).

The students' interactivities during problem solving are used to update Bayesian network with nodes and conditional probabilities representing how these actions influence the probability that the student is performing the task and utilising the knowledge components correctly.

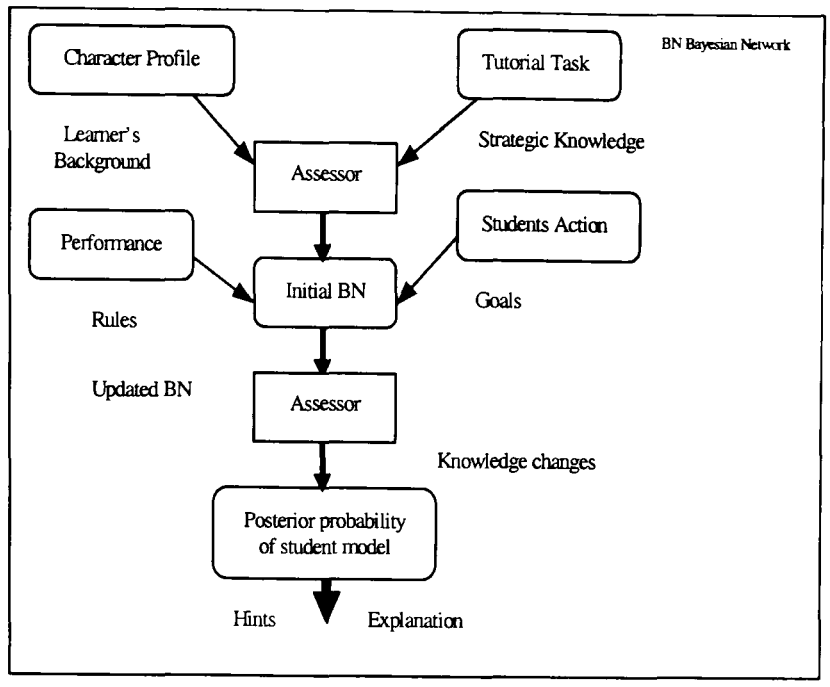


Figure 4.12 Structure of GeNisa Student Model

Nodes representing pedagogical tasks directly influence the probability that the student is deviating from the case scenario (rule-case node) in the student model. Nodes representing case scenario activities are performed through the user interface, which may influence the probability that the student is applying a correct analogy. Therefore, during the student's interaction with GeNisa, the system assesses the student's knowledge and understanding of the case scenario. Furthermore, the probabilities associated with rule-application nodes demonstrate that the student has correctly executed the case scenario without major derivations. Rule-case nodes with probability below a given performance level require immediate interventions. At the end of the tutorial session, the student's probabilistic rules are used to update the student model.

4.4.3 Tutorial Hints

The student's input and methods of problem solving are used to assess student actions and provide appropriate feedback and hints. The student model uses the pedagogical tasks to construct hints and explanations. GeNisa uses a hinting process (Hume et al., 1996) so that the student discovers knowledge by him/herself. Hinting is used to provide the student with a piece of information (Anderson, 1990), that can facilitate the recall of the facts needed to answer the question or complete an exercise (Gertner et al., 2000).

The student model is used for determining when and how to hint, and student responses to hints are used to update the knowledge acquisition module. Since there may be more than one pedagogical plan for tutoring a domain concept, the hinting strategy is closely related to the

pedagogical task or tutorial plan. Provision of hints may facilitate self-explanation, explicating the learner's line of thought, although, real-time interpretation of the learners' actions is essential for interactivity and for appropriate, and meaningful responses to the learner's needs. In order to facilitate hinting and to ensure interactivity, this research used a set of autonomous agent components for responding to different range of users' behaviours. Each agent can respond according to the context of interaction.

4.4.4 Reference Library

A reference library (depicted in Figure 4.13) is provided as part of the learning environment. This is used to provide references to materials that are relevant to the current domain and for enhancing the learner's domain knowledge. The reference library is represented as a Web-based reference library and spans the different area of simulation modelling and consists of examples of simulation modelling.

Exploring the tutorial task involves using a case scenario, which consists of a description of a hypothetical client's business and a problem area. A case-scenario-based approach is highly intuitive and can improve instructional quality. Students can solve the problems by either working in the "Practice", or in the "Tutor" mode. Furthermore, in the "Practice" mode, the agent monitors the student's sequence of problem-solving actions and interrupts only if a violation takes place. The practice mode allows the learner to take all the "learning initiatives" (Horvitz, 1999). This approach allows the system to behave more interactively with learners and provide the learner with direct control of the tutorial. Furthermore, it provides the learner with freedom to use his/her knowledge to practise problem solving.

The fundamental epistemological concept underlying this approach is that it is more beneficial for learners to "develop and debug their own theories than to teach them" (Wenger, 1987). In the "Tutor" mode, GeNisa guides the learner through the case and directs the learner through the essential task domain that must be performed. One advantage of this approach is that if the user cannot provide appropriate responses, the learner cannot proceed in attempting to solve the problem. Depending upon the instructional goals, GeNisa may highlight aspects of the case, suggest correct actions, provide hints and rationales for particular actions, reference relevant background material, and provide a contextual assessment. These actions are domain independent and can be used in most tutoring methods, e.g. coaching (Breuker, 1990).

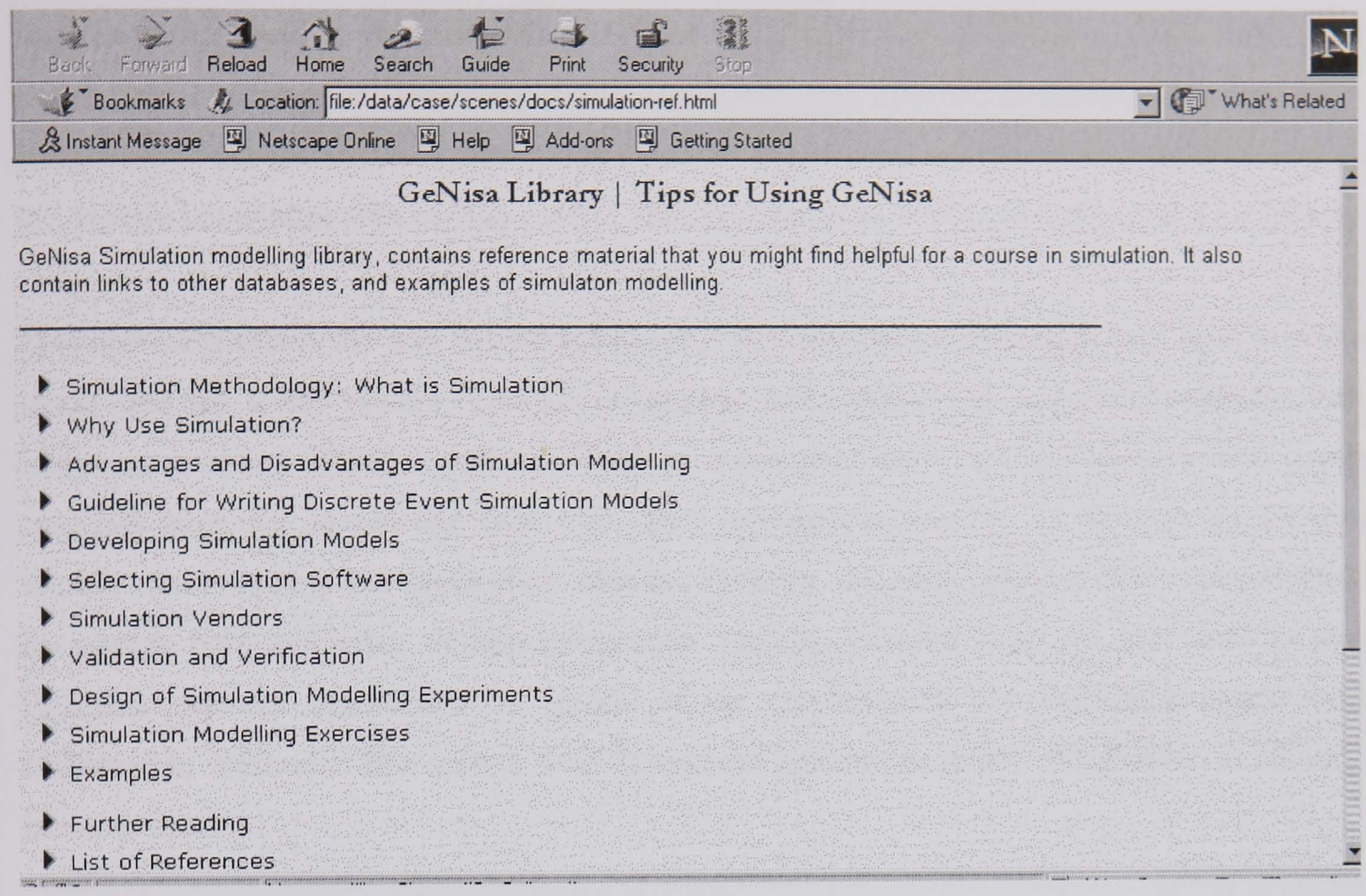


Figure 4.13 GeNisa Reference Library

Each tutorial task implements a Dynamic HTML interface and references a file as a resource. This approach allows each HTML page to be embedded in an EXE or, DLL, or even to be located on a server.

4.4.5 Knowledge Acquisition Module

The Knowledge Acquisition Module (KAM) uses user inputs, knowledge bases, and other information sources to guide during knowledge acquisition. The KAM is encoded as probabilistic rules and Bayesian networks (Pearl, 1988b), (Pearl, 1993), (Heckerman et al., 1992). They provide a methodology for capturing uncertainty in domain knowledge. Bayesian networks provide a graphical, intuitive, and computationally tractable means of capturing prepositional relationships in a domain, and can be used for such diverse applications as situation assessment, plan evaluation, and diagnosis (Friedman et al., 1997). This aspect of the research is grounded in previous work in AI and probabilistic rule induction from data (Friedman et al., 1997).

In order to enable a domain expert to build domain-specific probabilistic models, GeNisa's KAM uses Bayesian induction methods that permit the user to provide a very basic specification of the model that is close to his or her objectives and expertise. This specification (user inputs) will

include partial models (subnetworks and rule sets), relevance statements about input/output events, and explanations (inference chains associated with specific scenarios). The key to this framework is the utilisation of knowledge obtained from user interactions, the case scenarios and the domain knowledge (Heckerman, 1997), (Heckerman et al., 1992), (Friedman et al., 1997).

This approach allows the user to interact directly with the knowledge acquisition tool, providing inputs into the KAM system (tutorial interactions and explanations associated with the user's inference process). These inputs are used as constraints on a learning process that gathers evidence from the knowledge base and other information sources, to discover probabilistic models characterising the application domain. Training data are gathered from the foundation knowledge base and other on-line information sources. Learned rules are also used to guide Bayesian network discoveries by means of an extension of the knowledge-based model construction methods. The system then formulates queries to refine the models in an active learning process.

Explanations of a user's inference process for previously observed or hypothetical situations will permit the user to train the system through learning by demonstration. Expert models that are explicitly made for the purpose of training the system could construct the explanations.

4.5 A Tutorial for Simulation Modelling: An Example

Simulation modelling learning environment was designed and implemented in order to investigate the feasibility of the generic architecture learning environment. Simulation modelling domain was chosen because it consists of curriculum-based tasks that can be explored for computer-based tutoring or ITS (Paul et al., 1998). Furthermore, simulation modelling is particularly suitable for case-based ITS as it involves analysis, diagnosis and, it allows the learner to work interactively with real-life simulation scenarios (Atolagbe and Hlupic, 1998).

The simulation case scenario was implemented by using SimTutor class library, a discrete event-simulation modelling class library. SimTutor builds on the SimJava (Howell, 1997) class library. The SimTutor builds on the fundamental abstraction of the subsystem in SimJava to provide additional abstractions for simulation modelling tutorial (e.g. parts, workstations, conveyors, and routers). SimTutor class library includes classes for representing graphs, and animation and basic statistical analysis. All simulation classes were implemented in Java, and consist of essential classes for developing simulation for instructional purposes. This framework is similar to

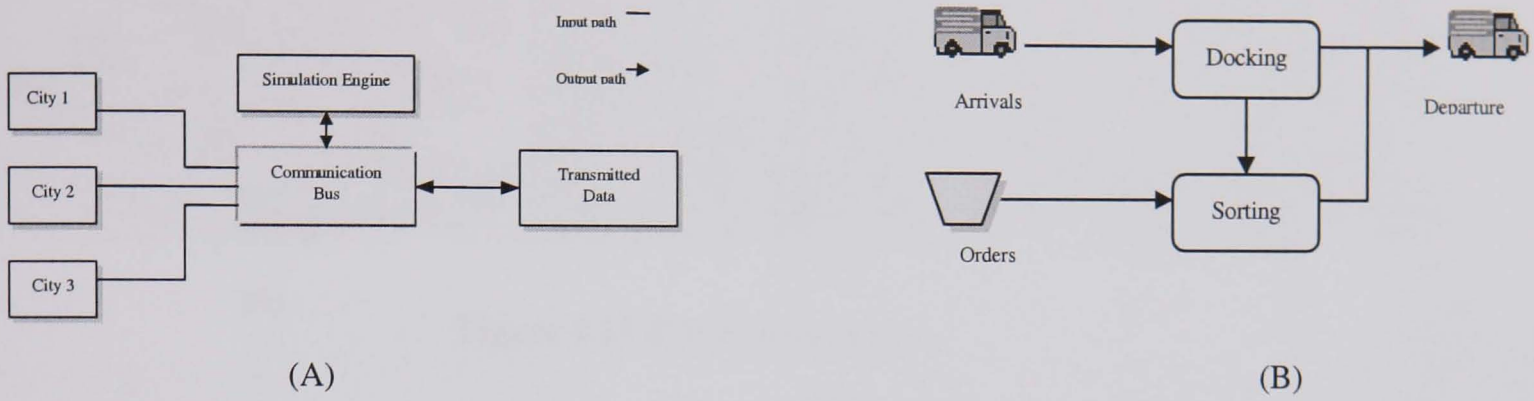
SimJava (Howell, 1997), Simkit (Buss and Stork, 1997), JavaSim (Little, 1997), and DEVS-Java (Zeigler, 1997). All these packages are based on object-oriented programming (OOP) (Adiga and Glassey, 1991). OOP paradigm is suitable for the discrete-event world-view formalism (O'Keefe, 1986), (Burns and Morgeson, 1988) because it facilitates modular design and simulation software reusability (Zeigler, 1991), (Mize et al., 1992). This approach allows simulation model to be developed without using a simulation package. Furthermore, the SimTutor package has been implemented in Java in order to support deploying application over the Internet. This approach allows simulation applets to be integrated within the instructional environment, and to support pedagogical activities.

Simulation model classes are directly instantiated against the "entities" they represent, or extend their classes. Entities were used as the main building block for simulation model development. The behaviour of an entity over time during a simulation is implemented through events. All events are represented procedurally as methods of classes and encapsulate the behaviour of the entities. The class library also contains several classes to represent various simulation activities, exhibiting different behaviours. Different entities are used for building the simulation case scenario and all entities are linked together by using a "port". The SimTutor package consists of the following class libraries:

Animation. Animation package consists of classes that can be used to illustrate a concept, for example, to illustrate an aircraft stability control or network of computer animation etc, which can illustrate the potential problem. Each animation class consists of parameter variables that can be manipulated by the use. Each frame can hold a piece of data from memory, with an associated address tag. Text boxes and buttons allow the user to control the simulation and change initial parameters. Entities and ports have their own icons loaded from graphical interchange files. The icons can be changed to represent the current state of the entity, and other entity parameters can be displayed as text. Messages passing between entities are displayed as squares, which travel along the connecting lines; the number attached to the square is the message tag.

Statistical Distributions. SimTutor statistical package can be used for generating Gaussian Normal and the Uniform distributions (U (0,1)), (Law and Kelton, 1991). The statistical package uses Java random number variate (mathematical function class) to generate different type of statistical distributions. For example, the statistical package can periodically collect data about every different entity, the maximum number of entities during the simulation, the mean, the standard deviation, and the variance of different entities.

The simulation example consists of three main subsystems: geographical service area, modelling and analysis module, and output analysis subsystems. Each of these subsystems consists of multiple components for graphical and data management. The execution flow of the simulation example is shown in Figure 4.14 (A). Each city sends and receives data (stored in the array) from across the network through the communication bus.



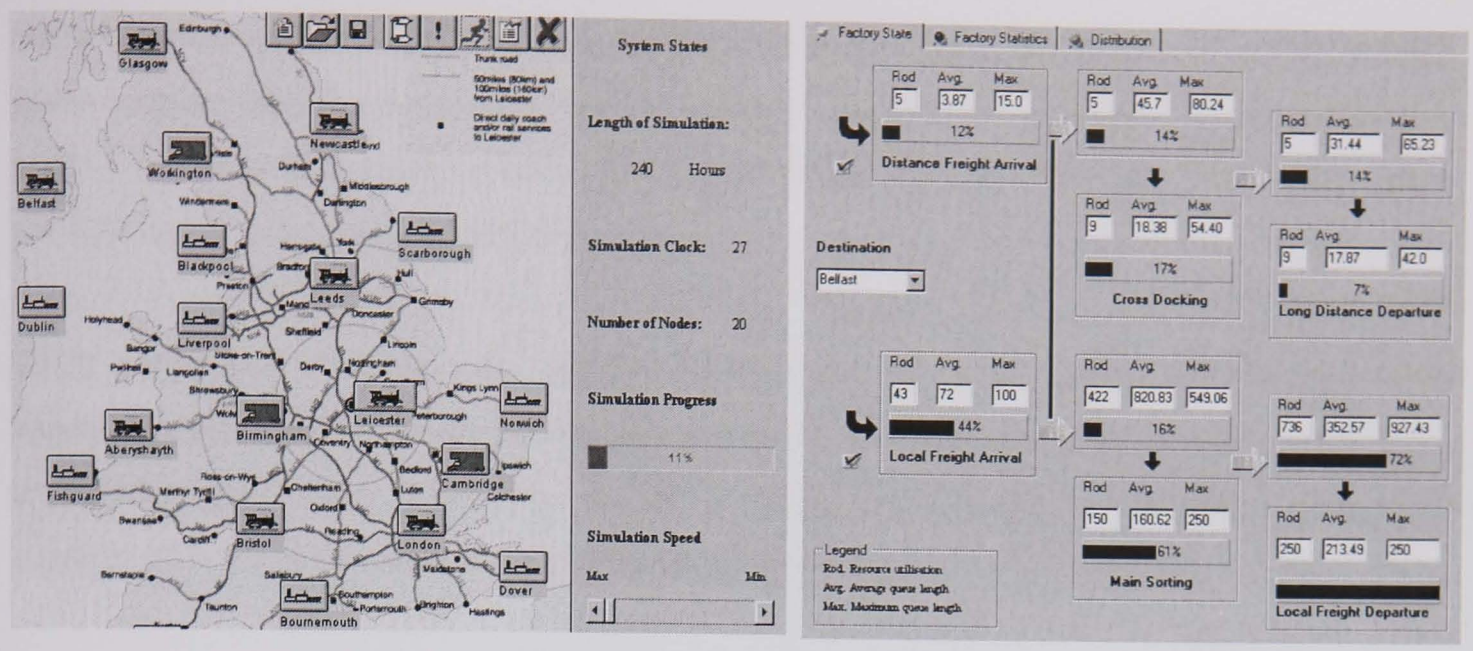
(A) (B)

Figure 4.14 Scenario Events Model/ Operations

Figure 4.14 (B) depicts the operational scenarios, which offer students an interactive environment characterised by a range of scenarios that exemplify the general factory logistics problems. Bulk shipments are processed at the docking area, and are pre-sorted into different product groups, by destination (local or national), or stored in the warehouse. After the pre-sorting stages, products are sorted again (secondary sorting) according to the haulage requirements and product type (e.g. finished products or components). Products are picked within a flow rack, and by pick-to-belt operation in the sorting area.

The communication bus is used for handling interactions between different cities and for transmitting events. The simulation engine controls the length of simulation and simulation time. At the end of each simulation cycle, statistics on the different parameters of the network, such as the cycle time, costs, transportation costs and service area traffic are generated.

An illustration of the simulation modelling applets is depicted in Figure 4.15. The system allows the learner to address some manufacturing distribution problems such as determining the ideal number and location of suppliers, transportation time, resource utilisation and forecast demand across the service area, as well as distribution methods.



(A) (B)

Figure 4.15 Simulation Applets

Applet (A) depicts the system simulation, service areas/locations, and the current simulation states, and Applet (B) depicts statistical distribution of a factory. Applet (B) represents other components that can be used to assign and edit parameters for each factory within the hypothetical service areas.

Each factory contains subsystems for estimating resource processes distribution (Rod) and components for local shop floor analysis. Consignment may be passed from one location to another. When a consignment is passed to the next location, it will either be held in a queue at the new location, or the centre will process the work according to predefined production floor parameters. The parameters can be customised for different simulation by means of lists and controls. Furthermore, consignments are presumed to be generated in uniformly distributed (U(0,1)) (Law and Kelton, 2000), (Pidd, 1998), (Paul and Balmer, 1992) manner due to fluctuation in demand. After each simulation, the system produces different statistical output for data analysis. For example a factory may produce the following outputs: number of consignments arrivals, number of goods dispatched, number of consignment being processed, average number of consignments in the queue and queues processing time.

One of the main outputs of the simulation is the cost of transportation. This research used a cost function (Daganzo, 1996), which expresses the total cost (in pounds per day) of the distance travelled by the trucks and the freight operations. This is based on the assumption that the time unit of the cost function is the day. The cost function is computed from:

$$Z_{td}\left(\bar{d} + \frac{k}{S^{y_2}} \right)\frac{q}{s_t} + Z_{ds} + Z_w.N(£/day)$$

Where S is the number of service area, Z_{td} is the truck cost per unit-distance; Z_{ds} is the multiple-drops cost per day; Z_W is the truck-waiting cost per day; d is the average distance travelled per truck; k is the local distance; q is the quantity of freight delivered in weight unit-freight in tons per day; and S_t is the capacity of the truck in tons.

These parameters accentuate the importance of production costs, which relate closely to the distance travelled by the trucks per day. Moreover, the cost function allows the user to learn how to manage the factory operations so as to maximise throughput and lower costs, with consideration for the demand and distribution capabilities.

This approach may enable learners to learn simulation adequately and efficiently by applying theoretical knowledge, and making comparisons, and by investigating the simulation model more of a real world problem (Paul and Balmer, 1993), (Davies and O'Keefe, 1992). This approach may also help the student to develop an understanding of a number of fundamental concepts in operational management such as transport and shop floor utilisation.

4.5.1 Student Activities During Instruction

During instruction, GeNisa uses the pedagogical agent to guide the learner dynamically through the case scenario, and refines the tutorial plans according to the student's needs and pedagogical activities. This approach allows users to use self-explanation in order to elucidate the case scenario, which enables them to question and repair their understanding (Chi, 2000), although, using self-explanation strategy may result in different levels of instructional outcomes. However, GeNisa has different levels of knowledge and components that can support different learning outcomes. Some of these components include:

i. *Conversation*. This component allows the student to interview the client by asking a series of questions on current and business practices. The client provides an immediate reply to all questions. The purpose of this is to teach the student the knowledge required for conducting analysis of simulation, and the needs to identify the client requirements before commencing analysis. Figure 4.16 depicts the screenshot for the conversation components and the client's responses.

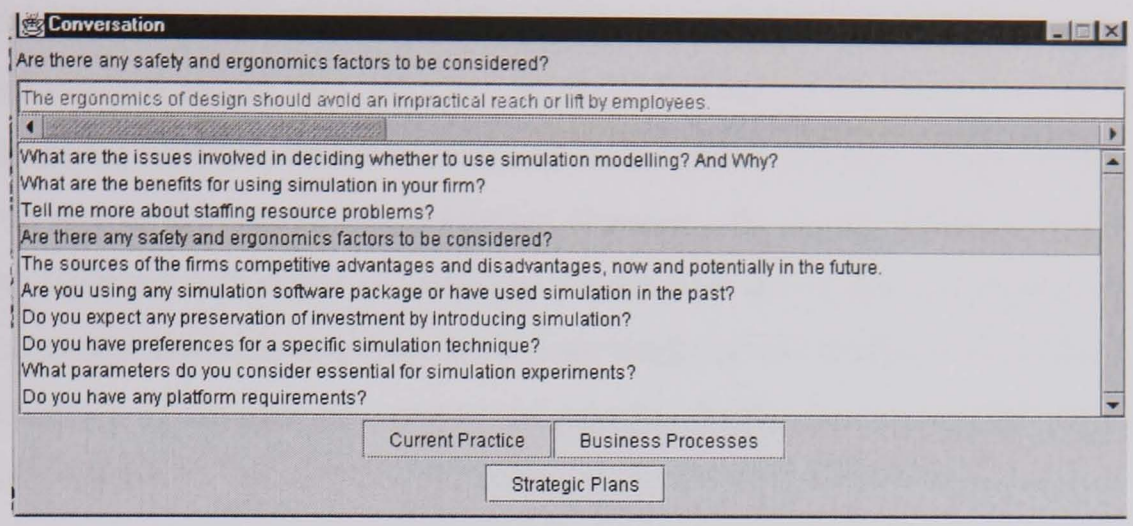


Figure 4.16 Client Interview

ii. *Case Diagnostics Tools*. Results obtained during the interview with the client are used to obtain the requirement and to further analyse the business problem area. This is shown in Figure 4.17. These applets provide detailed theoretical knowledge, which may be necessary before implementation. Furthermore, the case diagnostic tools allows the user to make instructional decisions and to self-explain some instructional tasks.

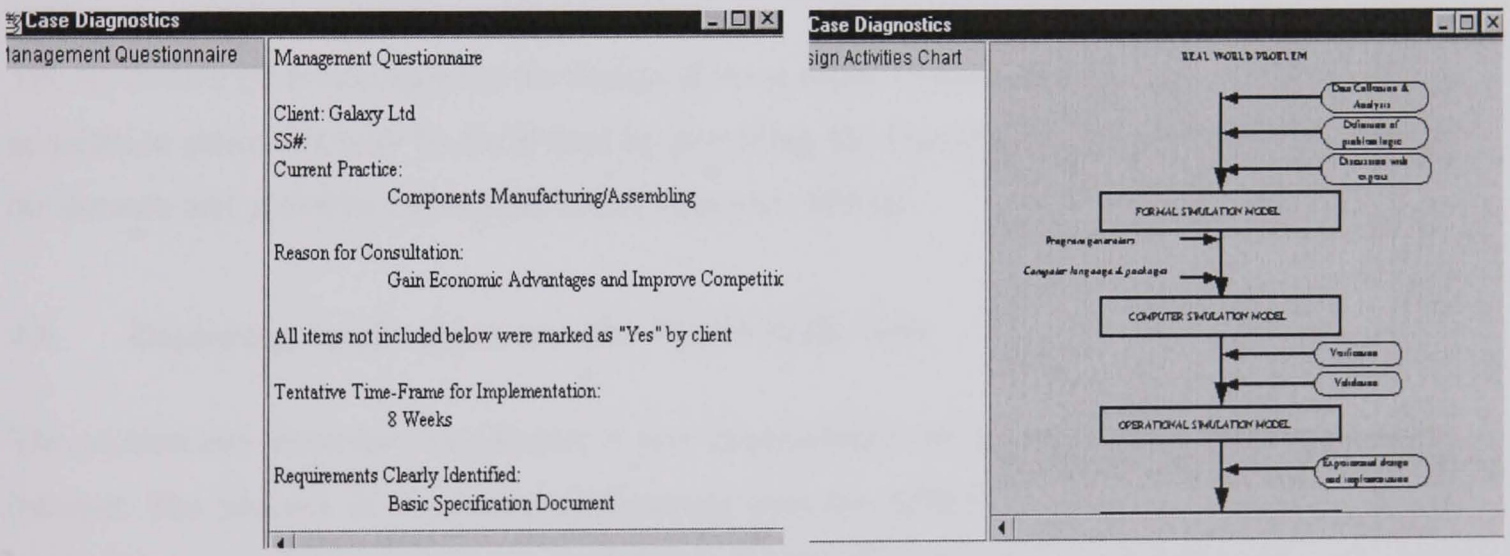


Figure 4.17 Case Diagnostic Tools

iii. *Performance Model*. The performance model uses the student model to compute and maintain an assessment of the learner's competence level of each pedagogical task. It interprets the s (A) s problem-solving action in the context of the cu (B) roblem and determines the type of feedback to provide. The performance model then updates the student model. When the problem-solving task is completed, the assessment module analyses the student's record and provides appropriate feedback. For example, GeNisa provides two types of pedagogical task assessment: (i) an evaluation of the student's

analysis of the case scenario, and (ii) evaluation of the procedures taken by the student. The performance model uses information about differential analyses, such as conformance to standard guidelines, and may comment on the users' analysis, or may compare an incorrect response to the correct one. Different domains will require different assessment modules, and feedback will differ accordingly. Figure 4.18 shows the quiz tool for testing student's knowledge during instruction. Immediate feedback and explanation is provided if a learner chooses a wrong response, and he/she may be directed to the reference library (discussed in this section).

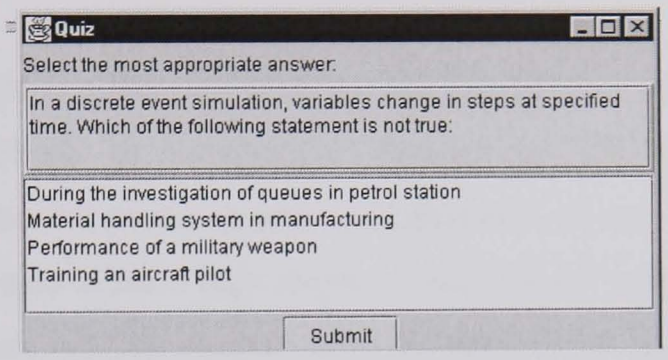


Figure 4.18 Quiz Tool

The epistemological rationale for the design of these tools is based on the premise that knowledge acquisition processes may be facilitated by providing the learner with detailed knowledge about the domain and problem solving methods (VanLehn, 1996a).

4.6 Deploying Applications over the World Wide Web

The architecture described in Chapter 3 was implemented for application deployment over the Internet. The process of deploying applications over the WWW involves the following: (i) the application to be deployed over the WWW is implemented as a Java-enabled applet, and (ii) it insures that users who want to run the application have the Uniform Resource Locator (URL) to access the proper HTML file, and the appropriate Web browser plug-in. The Web browser plug-in communicates with the Web-server by sending user requests through its HTML page.

The Web HTTP server communicates the user's requests to the Web-enabled applet on the application server. The Web-enabled applet communicates with the database for the data it needs on the application server. When the application is fully developed and ready to be built and delivered, it can be deployed as an applet for users to access via the WWW. The applet resides on the application server. When the user requests an applet, its files are retrieved from the server and placed in the temporary directory of the client's machine.

The general architecture has components (for example, client-side components) responsible for integrating with Java enabled browser and for communication with the server. The applet provides an interface to the server functionality and supports HTTP protocol. All anchors returned from the link to Web pages are encapsulated in JavaScript and inserted into HTML, causing the browser to call back the applet when a link is followed. Although this solution is platform-independent, current implementation is browser dependent, as the implementation requires the use of specific Netscape API to facilitate communication between JavaScript and Java applets.

Figure 4.19 shows the GeNisa client user interface and the main tutorial components. The GeNisa client permits multiple clients to communicate through an API with the server. The server manages each client connected to the system and maintains records of the application the user executed. This approach may allow a high degree of component independence as both the client and the server applications need to execute only the appropriate platform protocol in order to deploy application on heterogeneous environments.

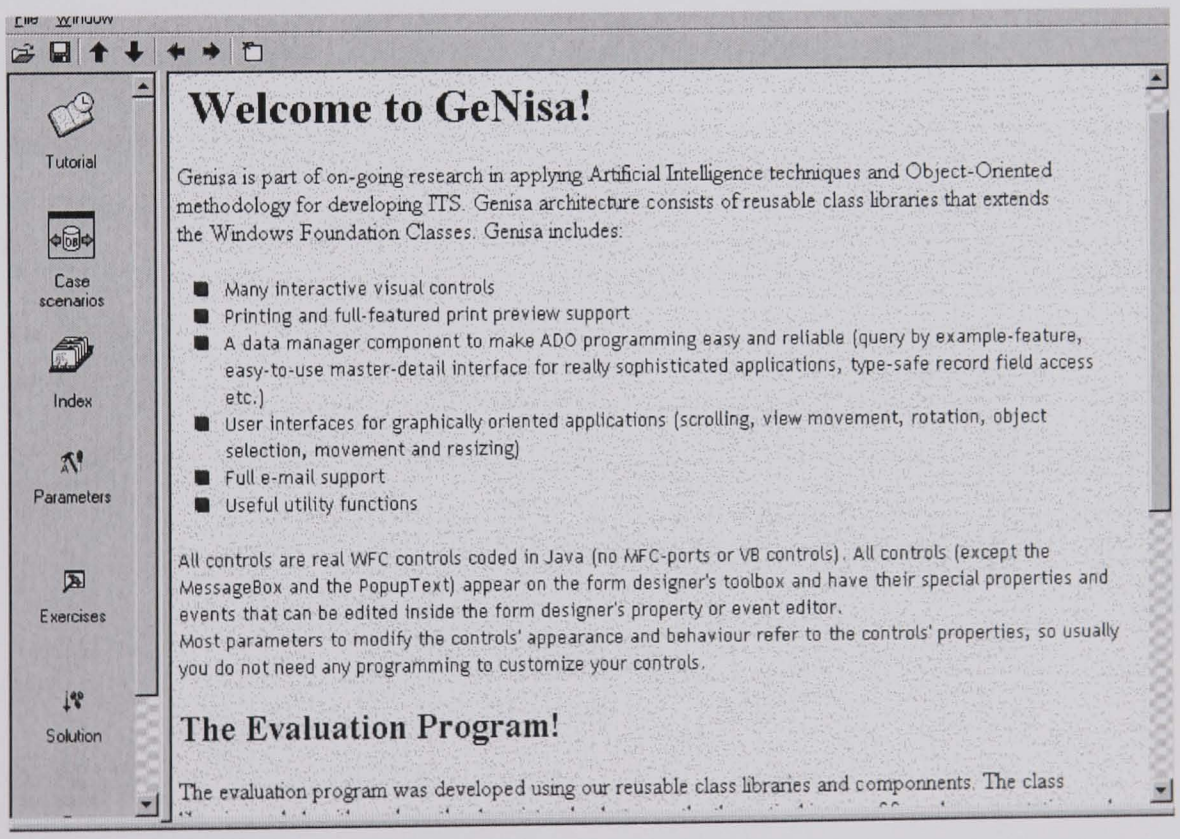


Figure 4.19 WWW Interface

4.7 Summary

This chapter has described the development and implementation of the GeNisa. The implementation of GeNisa was based on the conceptualisation of the generic architecture

described in the previous chapter. Object-oriented class libraries are used to manage the large quantity of code developed during the development of GeNisa. This chapter has analysed the structure of such class libraries that support the different range of ITS components. The overall systems architecture has been presented in terms of a heterogeneous collection of systems providing a wide range of application functionalities.

This chapter has also described an example of learning environment of the generic architecture for teaching simulation modelling. The tutorial uses a case scenario to guide the user through the instruction. The example also makes use of an animated pedagogical agent to enhance the interface within the learning environment. Different components that constitute the learning environment are discussed.

CHAPTER 5

EVALUATION OF THE GENERIC INTELLIGENT TUTORING SYSTEM ARCHITECTURE (GeNisa)

5.1 Introduction

This chapter presents evaluation of the design and implementation of the generic architecture discussed in the previous chapters. The evaluation is conducted by using formative evaluation (Mark and Greer, 1993), (Murray, 1993), (Legree et al., 1993), (McGraw and Harbison-Briggs, 1989), which is characterised by cycles of design, implementation, and evaluation (McGraw and Harbison-Briggs, 1989). The objective of this chapter is to appraise critically the work carried out in this study and to provide theoretical and empirical constructs for justification of the study, and to establish the significant benefits derived from GeNisa.

A questionnaire was used to evaluate the characteristics and performance of the generic architecture (Osgood et al., 1957). The questionnaire consists of bipolar adjectival pairs of opposite meaning and is associated with a particular construct on a five-point scale (Osgood et al., 1957). The construct used in this study were obtained from reviewed ITS and OO literature, usability guidelines (e.g. Nielson, 1993) and experience gained during the design and implementation of the GeNisa. This framework appears to be the most useful for measuring "affective" attitudes (Osgood et al., 1957) for the use of the components. It also offers the user flexibility and, according to Heise (1969), has been shown to be both reliable and valid.

A significant number of methodologies for evaluating ITS are currently available (Murray, 1993), (Self, 1992), (Self, 1999), (Twidale, 1992), (Mark and Greer, 1993), (Shute and Glaser, 1990), (Shute and Regian, 1993), (Siemer and Angelides, 1995). A variety of these methodologies and their respective studies were critically examined in order to identify the most effective paradigm for evaluating GeNisa. Furthermore, all these studies outlined a methodological approach for evaluating an ITS. However, none of these studies provides a list of criteria for the evaluation of ITS components. Also, McGraw and HarbisonBriggs (1989) argue that it is difficult to develop specific measurable criteria for knowledge-based systems, and criteria based evaluation is suited

for evaluating specific aspects of a system, where criteria can be specified and measured (Mark and Greer, 1993).

The evaluation framework developed consists of a list of component features, which covers core capabilities of the components in GeNisa. The component features were based on object-oriented software engineering and AI literature. This evaluation framework was used by evaluators for the assessment of each component and its functionalities. Some of the specific features which the criteria examine are: content, usability, effectiveness, navigation, portability, reusability, modularity, impact, effective use of media, and degree of difficulty/ease of use etc. The development of the questionnaire is based on the list of criteria. An example of the questionnaire is shown in Appendix D. The evaluation criteria provide a systematic and practical means for critically evaluating the effectiveness of the components features in GeNisa, because multidisciplinary, methods for evaluating ITS are inherently problematic (Hlupic et al., 1999), (Mark and Greer, 1993), (Shute and Regian, 1993). The generic architecture is also examined in by using style guidelines and usability heuristics (Nielsen, 1994), to evaluate performances of GeNisa, and to highlight potential usability problems (Nielsen, 1993).

The design and implementation of the generic architecture has been evaluated theoretically, by using a software engineering approach (Sommerville, 1996), (Booch, 1994). This involves considering the requirements of all other possible applications, and by establishing appropriate relationships between the architecture's heterogeneous component properties.

The rest of this chapter is organised as follows. First, evaluation objectives are presented, followed by detailed discussion of formative evaluation of GeNisa. Finally, an analysis of the evaluation results is presented.

5.2 Conducting Evaluation of the Generic Architecture

This section will evaluate the design, implementation and performance of the generic architecture. The aim is to highlight problems that may arise from this study and to describe different issues and benefits of the GeNisa architecture. The performance and evaluation of various components of the generic architecture also provides a source of data used to analyse GeNisa and its components. Furthermore, it is also used to identify a set of components, which form the basis for the improvement proposals for further development of ITS (discussed in Chapter 6). This section firstly discusses the evaluation objectives of this research, which is

followed by conceptual evaluation of the generic architecture. The evaluation of the generic architecture may help identify some limitations of the design and implementation of GeNisa discussed in previous chapters.

5.2.1 Evaluation Objectives

Within the evaluation objectives this research will endeavour to appraise each component of the generic architecture critically and to examine the design and implementation framework discussed in Chapter 4. This will illustrate how the requirements for the generic architecture discussed in Chapter 3 are supported by the implementation formalism. The objectives are: (i) to evaluate the usability of GeNisa components and to identify problems that might affect portability and reusability; (ii) to examine research findings which could form the basis for the proposed methodology for ITS (discussed in Chapter 6); (iii) to critically appraise the design and implementation of different components of the generic architecture, and their functionalities. This is done by comparing GeNisa to traditional ITS, and by identifying possible areas for enhancement of the components. This is important because the more generic component features can be made, the easier it will be for use in different domains without any decrease in components interactivity/functionalities; and (iv) to examine the usability of the user interface (Nielsen and Mack, 1994) and to ensure that the various interactivity requirements (discussed in Chapter 3) are adhered to and are consistent (e.g. the use of appropriate metaphors and direct manipulation for intra/inter component navigation). Each of these objectives is explicit in the evaluation of the generic architecture and its components.

5.2.2 Evaluating the Generic Architecture

GeNisa combines an object-oriented approach with AI techniques to enhance the creation of interactive ITS and an automated method to aid users in managing the knowledge development processes. As discussed in Chapter 4, GeNisa utilises case-based reasoning techniques, which provides users with means of sharing, exploring and adapting different cases during instruction (Atolagbe and Hlupic, 1998).

Most of the components used within the GeNisa architecture rely on the behaviour and contents of other components. Therefore, the integration of different components may introduce bugs into the software (Szyperski, 1998). It may be tedious to test all possible combinations between components of the GeNisa architecture. However, interoperability problems are inevitable. It is

unfeasible for applications that can be independently extended by the end-user, and of course for a generic architecture.

It is to be expected that an object-oriented approach alone will not make a major difference to the knowledge acquisition problems in ITS development. The object-oriented programming paradigm provides developers with the capabilities for managing large bodies of code through structured modular decomposition that simplifies code generation and maximises the possibilities for reuse (Booch, 1991). However, the effectiveness of the paradigm in providing support for large-scale software development through modularity and reuse has not yet been proven (Biggerstaff and Perlis, 1989).

Numerous inter-component consistency constraints exist between different components used for the implementation of the GeNisa. These are checked and preserved by the components contained in the development environment. Each class, in all the tools, must be refined in terms of a class definition and an implementation method. If a relationship between classes is changed, the respective methods and constructor declaration are consistently changed in the class definition. Similarly, the integration of the XML tools allows component attributes to be easily retrieved and edited. Furthermore, the matching of method signatures of the class definition and their corresponding definitions in the implementation is ensured. The GeNisa environment preserves the inter-component consistency constraints that exist between its classes and class interface definitions. For example, when a developer creates a new project, a respective class definition is automatically created. Moreover, inheritance relationships are introduced depending on the kind of project or tool.

5.2.3 Interoperability and Reusability - An Appraisal

At the conceptual level, interoperability and reusability are aided by shared ontologies representation (Musen et al., 1995), (Gennari et al. 1994). Ontologies are presented as task structures, e.g. task hierarchies that define abstract concepts and the relationships between them. The essential features of each component are abstracted into their constituent features, which may be shared with other components and reused.

The events communication manager implementation is reusable with other components, although, in order to implement component-specific functionality, additional event process classes may have to be added. The component event communication processes are essential for

interoperability and reusability. Therefore the implementation of events communication manager may facilitate the data interchange, addition of new components modules, and reuse of components across platform (Sommerville, 1998).

The object oriented paradigm used for the design and implementation not only contributes to the reusability of components (Booch, 1994), (Sommerville, 1998), but also allows the developers to extend components classes as required. This framework allows a complex component specification to be structured into manageable modules according to their classes. Furthermore, the component structure is supported not only for their classes, but also on a more coarse-grained level for subsystems in the entity relationship notation (Booch, 1994).

5.2.4 Evaluating the User Interface

Different components of the development and instructional environment reflect the general systems functionality. The design and implementation of the user interface is based on (i) identification of interactive strategies features for use in both authoring and instructional environments (discussed in Chapter 3), (ii) development of suitable case scenarios, pedagogical and tutorial strategy and Bayesian inference rules, and (iii) implementation of interactive features for each component. Interactivity is based on direct manipulation dialogue (Shneiderman, 1998). Each component executes a set of possible tool commands that are applicable to the current user activities and presents the components in a context-sensitive menu to the user.

The importance of providing good user interfaces for interactive ITS systems is widely recognised (Brusilovsky, 1996b), (Akpinar and Hartley, 1996), (Schwier and Misanchuk, 1993), (Waterworth, 1992), (Cawsey 1992). A quality user interface must provide consistency, reliability, self-sufficiency, ease-of-use and interaction flexibility (Nielsen, 1994), (Shneiderman, 1998). The design and implementation of the GeNisa is based on these studies, which are used to provide consistency of the GUI components such as icons, buttons, menus etc. Also, components event parameters are passed to other components in the same way during execution (Sommerville, 1998). The GUI is examined by employing heuristic evaluation (Nielsen, 1994) and questionnaire methods. Interaction flexibility (Sim, 1994) provides the means for addressing the multiplicity of ways the user and system exchange data. This will help to examine the consistency of the user interface, to appraise the usefulness of the metaphors used in GeNisa, and, finally, to postulate recommendations for enhancing the quality of the user interface in ITS.

5.2.5 Evaluating the Implementation

The objective is to examine the processes used during the implementation of GeNisa. The implementation of GeNisa prototype (described in Chapter 4) is based on the class library, abstracted into different packages to enhance its portability. Each component is organised according to its composite elements, and templates (parameterised classes) have been used to abstract commonality between similar classes. It has a very distinct separation between different component modules and with a very defined interface between them. Each of these components implements a high-level functionality (e.g. student's diagnosis, proving hints, critiquing user, etc.). All actions invoked by the user are accomplished through an interface, which invokes the required application. The class library implementation framework allows GeNisa components to be reusable across the domain, which may shorten component development time and support evolving component development. When GeNisa class libraries and its interface are used together, the two components may be configured to run on different machines - due to the carefully defined interface between the two. Moreover, the graphical user interface is implemented within Java Abstract Windows Toolkit (AWT) library, which allows GeNisa to be configured as a stand-alone program, as a network environment, and as a plug-in component to World Wide Web. All classes have been designed to conform to object-oriented programming and design patterns paradigms (Gamma et al., 1995), (Pree, 1995), (Buschmann et al., 1996) to increase the comprehensibility of the program.

The language implementation and design used object-oriented paradigm. Besides, the class library approach has explicit specifications for the syntactic and semantic connections between entities in the architectural and components method (Liebernau and Backhouse, 1990). This framework helps separation of the system's functionality into small modules, and helps decomposition and implementation processes (Booch, 1994), (Sommerville, 1998).

5.2.6 Assessing the Design of the Generic Architecture

The design of the generic architecture requires that the software and hardware are addressed for scalability, reusability and portability. The assessment of the generic architecture is presented in terms of existing technologies and how issues of portability and reusability have been addressed. Nevertheless, this assessment is based on comparative examination of the generic architecture based on contemporary literature.

The design of the generic architecture drawn on standard ITS architectural (Wenger, 1988), (Ohlsson, 1987), AI and object-oriented approach (Booch, 1994). This approach supports rapid

development of wide range of components of the architecture (Murray, 1999). However, the overall generic architecture is presented as a heterogeneous collection of components providing a wide range of application functionalities. The detailed design of each component of the architecture is analysed with cognisant views of domain and platform requirements, and the underlying theoretical principles that lead to an effective object-oriented implementation. The *design pattern* (Gamma et al., 1995), (Pree, 1995) has been used to design components framework for the architecture. The design patterns provide implementation-independent reusable design descriptions of the components (Gamma et al., 1995), (Pree, 1995), (Buschmann et al., 1996). They also allow components to have unique names, attributes and methods, which describes the components behaviour. This framework provides components' modularity, fosters heterogeneous reuse by allowing components to be replaced by newer ones without modifying the other components. Also, each component can be developed separately. The modularity of components should make transferring the tutoring environment to a different domain easier.

The rationale behind this evaluation is to examine the generic architecture conceptually, in order to highlight the main ideas underlying the design approach; also, to examine the constructs used for implementation of the generic architecture with respect to components-based development and reusability.

The methods represented in the class libraries conform to the Component Object Model (COM) (Box, 1998) approach, which allows the components to be run on any virtual machine. COM framework allows a language-independent binary standard for component-interoperability and allows components to be used for different applications and across platforms (Box, 1998).

The generic architecture provides design alternatives and default components for applications development, critiquing mechanism, visual design of components, and a wide range of event processes required by different components.

Each class method that has been defined in a class interface has been implemented in the respective class. Since components class libraries are developed for component reuse, they must be accompanied by components that enable users to reuse classes from the library. Apart from the design that identifies the different dependencies of classes, developers require an Application Programming Interface (API) of the functionality provided by the public methods of a class (Gosling et al., 2000), (Arnold et al., 2000). This API is defined in the Java documentation and it includes a description of each method and the fields.

5.2.7 Informal Evaluation

Informal evaluation was used to complement other evaluation methods discussed in this chapter. This approach was used to obtain assessment of the overall architecture from ITS developers and learners. Informal evaluation provides incremental enhancement of the architecture and provides ways of testing the functionality of the components before development (Twindale et al., 1992). This informal evaluation is generated mainly from enquires and from feedback obtained from different conferences and published papers (Atolagbe and Hlupic, 1996; 1997a; 1997c; 1998). Some informal enquiries are presented in Appendix G. This evaluation suggests positive evidence for the effectiveness of GeNisa architecture.

5.3 User Trial

Eighteen users representing four groups participated in the evaluation (depicted in Table 5.1). The groups of users were used to elicit feedback from GeNisa development and instructional environments, and to examine GeNisa's performance, usability, portability and to illustrate some functionalities of these components. The evaluators tried out GeNisa within a group of users (Tessmer, 1993), (Monk et al., 1986) and completed an evaluation questionnaire (copies of the questionnaires are provided in Appendix D). This feedback provides usability assessment and helps to identify areas for further improvements of the GeNisa. This evaluation does not seek to evaluate users competency in using GeNisa, but to verify that the components suggested are beneficial for both development and instructional activities.

5.3.1 Questionnaire

A questionnaire was developed to assess user perceptions of the usefulness of the GeNisa, to determine the users' usability assessment, and to measure the effectiveness of components in GeNisa in terms of implicit criteria. This approach helps identify areas of improvement of the components, and ensures that all the different tools in the generic architecture are adequately suitable for the domain and match the needs of the users. The questionnaire was based on evaluation criteria (described in Appendix C). Returned questionnaires were collated and analysed. The results of the evaluation are discussed in Section 5.3.5.

The questionnaire included two types of questions: Likert scale questions, and open-ended questions. The Likert scale questions asked evaluators to rate different aspects of the GeNisa on a five point scale. The points on the scale included *strongly disagree, disagree, undecided, agree,*

and *strongly agree*. The short answer section contained questions about positive features of GeNisa, and other questions addressing the GeNisa's effectiveness.

5.3.2 Evaluation Criteria

The majority of the evaluation criteria were drawn from different ITS literature, usability guidelines (e.g. (Nielsen, 1993), (Shneiderman, 1998), (Constantine and Lockwood, 1999)), OO and AI literature (e.g. (Steels, 1993), (Steels, 1990), (Drenth and Morris, 1992), (Chandrasekaran, 1988), and experience gained at various stages of the design and implementation of GeNisa. Some of these guidelines are specific rules that define a given window system (look and feel), others are more general-purpose heuristics. These guidelines were used in the development of the evaluation criteria (shown in Appendix C).

Based on the literature, this research proposed a taxonomy of criteria for evaluating ITS. The evaluation criteria centres on a coherent approach to evaluation that attempts to unify disparate evaluation techniques for ITS (Mark and Greer, 1993), (Shute and Regian, 1993). Instead of advocating a methodological technique, this research uses evaluation criteria to yield unification. This approach may allow ITS evaluation to be conducted so that it intuitively fits a given context, and is applicable to a broad range of domains. This is important in order to examine component functionalities in a wider context (Steels, 1993). Furthermore, the evaluation criteria may be used for explicit formalisation of reusable components and their underlying data structures.

5.3.3 Evaluators

Eighteen users representing four groups participated in the evaluation. The four groups and the percentage of users are shown in Table 5.1. Students with different backgrounds and levels of education (undergraduate students, MSc students and research students) and institutions (ITS research centres at other universities, ITS interest groups, and corporate organisations) were involved in the evaluation. The participants were randomly selected according to the following criteria: (i) for groups A and B subjects had to have experience of using ITS, (ii) the company has an information technology development department with permanent employees (this helped to eliminate temporary/agency staff, who may not be familiar with ITS development), (iii) staff have experience of developing either ITS or CBT, (iv) sufficient time had to be allocated for staff to conduct the evaluation.

Table 5.1 Description of Evaluation Groups

GROUP	DESCRIPTION	NUMBER OF EVALUATORS	PERCENTAGE (%)
A	Final year undergraduate students and MSc students	8	44.44
B	Research assistants and research students	4	22.22
C	Academic Staff and HCI Experts	3	16.67
D	ITS/Software Developers	3	16.67
Total		18	100

The evaluators with different backgrounds were chosen to increase the sample size and to obtain more diverse subject viewpoints and enhance validity of the findings. It is expected that this approach will help highlight the usability of the tools provided in the GeNisa and to identify any potential problems and design faults. Evaluators were given 90 minutes to complete the tutorial and a further 60 minutes to complete the evaluation questionnaire. Instructions were given to ensure that subjects knew how to use a GeNisa environment before beginning the experiments. Learners were also surveyed to elicit feedback about using GeNisa. The participants were asked to fill out the questionnaire after reviewing GeNisa, and to return the questionnaire and their written comments to the researcher.

5.3.4 Results and Interpretation of the Questionnaire

Evaluation criteria were quantified using a Likert-type scale, and items were tested for readability using an internal consistency method (Cronbach's Alpha coefficient, 1990), which yielded reliability coefficients of 0.99 and 0.96 for negative and positive items, respectively. These values were higher than the 0.80 criterion, which is regarded as internally reliable (Bryman and Crammer, 1997). An estimate of concurrent validity was measured using Pearson's product-moment-correlation coefficient, which yield a Pearson's correlation value of 0.92. The purpose is to ensure that scores obtained from one group of criteria are independent (not influenced by scores from other criteria) and thereby improve the validity of the scores. The results of the overall performance scores are show in Table 5.2, which illustrates the percentage score for each evaluation criteria. The percentage score was obtained from the analysis of questionnaire returned by evaluators. The mean percentage score shows that 84.45 % of evaluators think that GeNisa provided all the functionalies and components for ITS, and satisfies the need for portability and reusability. Several features score a very high response, which indicate that the feature is adequately represented and satisfies the evaluator's need. For example, suitability for courseware authoring/development scored a 88.88% response. More detailed evaluation results are provided

in Appendix E. Also, the domain-independent student model, the inference engine, the use of case scenarios, and the ability to deploy courseware on heterogeneous platforms scored very highly amongst participants.

It was noted that none of the responses suggested a lack of satisfaction with any of the components in either the development or instructional environments. The only comment that touched on this was "they're the same as CAI", which could be interpreted as a reflection of a role overlap between other components/or CAI.

As evaluators were not asked to contextualise their experience of ITS development, the source of their views may be assumed to be a result of direct experience with using GeNisa. A positive comment might reflect satisfaction with GeNisa and conversely, it is possible that their comments might relate to an idea of how they would like the component to be, rather than being a reflection of the actual functionality.

Table 5.2 Performance Score

Evaluation Criteria	SCORE (%)
Suitability for courseware authoring/development	88.88
Components implementation methods	84.67
Unique features and functionality	84.05
Module representation	77.64
Difficulties encountered	93.33
Overall design and look of the user interface	86.66
GeNisa performance	86.66
Components representation	83.77
Error tolerance	83.33
Reuse and portability	85.74
Learning environment	84.18
Mean Score	84.45

Evaluators could not clearly identify a problem area in GeNisa, this is indicative of a low weighted score. It could be inferred that the evaluators might not have fully exploited all the components in GeNisa, and were only able to suggest vaguely their experiences in ITS development. The lack of clarity mirrors the difficulty experienced by ITS developers themselves in articulating their roles and functions by using ITS components (Self, 1990), (Murray, 1999).

It was assumed that some respondents (developers) thought that GeNisa only worked in a domain. It is also recognised that participants have an ever present risk of bias, with them replying to questions in a way that they think the researcher wishes to hear (Borg, 1981).

In an attempt to reduce bias, the subjects were randomly chosen and were reassured that there was no risk involved to their equipment and confidentiality would be maintained. Also, evaluation was conducted at different sites with a group of three to four participants. Three sites were visited but the other companies declined to participate or had no staff with experience in developing ITS. However, some companies indicated that they were very interested in the evaluation, but could not commit the time.

It is, therefore, interesting that there were many similar responses, suggesting some consensus among the subjects about the functionalties provided in GeNisa, for example, the number of references to automated student model, student assignment, quizzes, hints, courseware generation and so on.

The evaluation emphasised a number of design principles, which provides a key to the usability of GeNisa. These include:

i. *Consistency*. The components should check for consistency between components of the same and different types, be able to visualise inconsistencies or even automatically preserve consistency during changes. Too many of the windows looked alike. This may create confusion about which menu was being viewed and about the functionality of different menus. Therefore, menu options should appear only if they are selectable. Also, there is a need for good screen management. The majority of the participants believe that consistency is an essential requirement of ITS.
ii. *Components Implementation*. Navigation mechanisms, and dialogue components permit a user to select displayed components by pointing and by using direct manipulation. Also, clearly marked "help" facilities, pop-up menus, etc., which are applicable to the user's current activity may improve usability. Reusable components such as Wizard, Design Assistant, Pedagogical Agents are modular and portable. About 84.67% of users believed that components implemented in GeNisa were good for authoring processes.
iii. *Interactivity*. Users goals, preferences and knowledge of their activities should be used throughout; the system should be easily adaptable to the needs of the user (Brusilovsky, 1996). Therefore, the system should provide good error messages for different tools at different levels of user. For example, error message such as "no class found", does not help

the user. The error message should be linked to a help file. Also, an indication that a selectable mode is active should be clearly displayed to the user.

Sellen and Nicol (1990) explained that the users make mental maps to help them navigate. Therefore, the system should make these maps explicit, thereby reducing memory load (Sellen and Nicul, 1990). Also, the system must provide the user with facilities to navigate through the system without getting lost. Participants overwhelmingly supported the interactive features and overall design and look of the interface. A sample size of 86.66% of participants agreed that GeNisa was very interactive. The sample also indicated that participates would reuse GeNisa.

iv. *Learning Environment.* Overall, the participants were positive about the components provided in the learning environment. They found the system easy to use and the pedagogy content was appropriate. Participants overwhelmingly indicated (84.18%) that the hints, quizzes and case scenarios were helpful during instructional activities.

By separating contents knowledge into domain knowledge and problem solving methods with a different reasoning subsystem, reusability and shareability of the components is promoted (Musen et al., 1995). The quizzes provide means for monitoring and assessing students' learning activities and are used to evaluate progress during instruction. This approach may help to direct users learning. Self-explanatory feedback and self-pacing provides added support for an inexperienced user. Anderson (1990) postulates that learners need to have immediate feedback in order to learn efficiently. GeNisa provides immediate domain-specific feedback based on user activities.

v. *General Comments.* The evaluation was designed to evaluate aspects of the design and evaluation criteria. Therefore, user responses were focused purely from the user's point of view. User comments and ratings were grouped as shown in Table 5.2. The lack of criticism evident in the users' responses showed that much of the system components were transparent. In general, some users stated that they found GeNisa as potentially helpful in their development activities. The questionnaire provided a range of knowledge and beliefs about the roles and functions of GeNisa. Analysis of these responses also suggests that the feedback reflected a more positive set of perceptions of GeNisa than might have been anticipated.

It was notable that none of the responses suggested a lack of satisfaction with any of the component functionalities that were offered. This may be attributed to the ease of using the components in GeNisa.

The generic architecture consists of a number of relatively independent component modules that can operate across platforms. Each component of the architecture, together with its class library, can be reused with different applications. For example the user interface components, the instructional environment and the inference engine, can be reused on heterogeneous platforms, and were perceived by the users to represent a fairly accurate representation of the components required for developing an ITS.

5.4 Heuristic Evaluation

In addition to the formative evaluation, heuristic-usability evaluation (Nielsen, 1994) was also conducted. Kantner and Rosenbaum (1997) postulated utilising heuristic evaluation during formative evaluations to avoid making errors. Heuristic evaluation was also carried out in order re-examine the usability of GeNisa, and to generate specific recommendations for improvements. Heuristic Evaluation (Nielsen, 1994) is a framework of usability evaluation, where an expert finds usability problems (Nielsen, 1993), by checking the user interface against a set of supplied heuristics (Nielsen, 1994) and make recommendations. The output of this evaluation will complement the formative evaluation and may provide more specific proposals for improving usability of GeNisa.

Many different types of user interface guidelines have been proposed, such as (Mayhew, 1992), (Brown 1988), (Apple, 1993). Some of these guidelines are too vague, and difficult to apply (Mosier and Smith, 1986). However, there is an increasing literature on different usability guidelines such as (Constantine and Lockwood, 1999). However, this research employed usability heuristics developed by Nielsen (1994) because the evaluator assesses usability problems by following a set of supplied heuristics. Usability inspection is a theory-based user-interface usability evaluation that breaks down an interaction into detailed steps and evaluates each step according to the Nielsen (1994) criteria. This evaluation process is generally concerned with usability of GeNisa, and helps to identify component's features that need to be improved.

The ten guidelines, suggested by Nielsen (1994), used for the heuristic evaluations are: (i) visibility of system status, (ii) match between system and the real world, (iii) user control and freedom, (iv) consistency and standards, (v) error prevention, (vi) recognition rather than recall, (vii) flexibility and efficiency of use, (viii) aesthetic and minimalist design, (ix) help that users recognise, and diagnose, and which enables them to recover from errors and (x) help and documentation.

5.4.1 Procedure Used

Participants were chosen from groups B, C, and D, as shown in Table 5.1. These groups were chosen because of their expertise, and it was believed that they could be able to provide more detailed feedback. Participants were familiar with software engineering toolkits (e.g. UML) and had object-oriented design experience. Participants had to have experience in using desktop applications and had to be familiar with using standard user interface elements such as dialogue, standard Wizards, tool bars, tree widgets, and direct-manipulation. Questions used for the heuristic evaluation and observation log are provided in Appendix F.

The evaluators were given guidelines for using both the development and instructional environments before performing evaluation. This approach was chosen to ensure that the participants could carry out different activities that required manipulating the graphical user interface. The participants were asked to carry out the tasks in the exercise (shown in Appendix F) and comment on the usability of the different components used. Participants were asked to evaluate the usability in accordance with the principles set out in Nielsen's heuristic evaluation (1994), and to comment on both the positive and negative aspects of the usability of the design.

Participants were urged to write down any situation encountered such as shortcomings in the user interface and suggest ways for improvement. Participants were given a predefined form to write down any observations/suggestions for analysis and discussion (Preece, 1994).

5.4.2 Results of Heuristic Evaluation

The ten guidelines postulated by Nielsen (1994), described in Section 5.4 provide a framework for the usability heuristic evaluation. The participants reported twenty points that are unique to GeNisa. The essential points were fed back to the original design. The results of the evaluation can be summarised and classified under the ten guidelines suggested by Nielsen (1994). The results are:

i. *Visibility of System Status.* GeNisa keeps all option menus and components for a given task visible, and allows direct manipulation (Shneiderman, 1998). For example, the system greys-out menu/toolbar component options that are not required for specific applications. The system uses visual indicators to give cues on how to use the tool.
ii. *Match Between System and the Real World.* Unfamiliar phrases must not be used in window titles and buttons. The system's careful choice of icons provides a quick way on how to use

a tool. The users may misinterpret some icons; therefore, all the system's icons relate to the selected tool.

iii. *User Control and Freedom.* The user should initiate and control actions of all components. GeNisa uses a modeless (Dialogue or Windows function that do not require the user to take an action before switching focus) dialogue to provide an "escape route" and clearly marked "redo" and "undo" tools for frequently used components.

iv. *Consistency and Standards.* All the components, pop-menus, toolbar, buttons, are consistent. The system provides good screen management for all applications. The system's use of colour and the active/inactive buttons, control bars, etc., are consistent.

v. *Error Prevention.* An indication that a component is selectable is clearly indicated. Active mode is provided for relevant tasks and the icons are clearly displayed to the user.

vi. *Recognition Rather than Recall.* GeNisa used prompts appropriately and the prompts are relevant to current task. The Design Assistant and the learning environments provide appropriate prompts on current task and at different levels of the system. This approach supports recognition and allows users to recall completed tasks, thereby reducing the "memory load" of using the components (Shneiderman, 1998).

vii. *Flexibility and Efficiency of Use.* GeNisa provides shortcut menus and direct access to essential tools on the toolbar, and on the graphical user interface. GeNisa allows users to tailor frequent actions and is customisable for different use.

viii. *Aesthetic and Minimalist Design.* The system uses only descriptive titles in buttons, toolbars, images/icons and dialog boxes. Each icon is used for a specific task only. There are no technical terms that may confuse the user or that could easily be misinterpreted.

ix. *Help that Users Recognise, Diagnose, and Enables them to Recover from Errors.* The error messages provided by the system are meaningful and content specific. Current implementation of system does not provide error codes.

x. *Help and Documentation.* Basic help and documentation is provided. GeNisa is intuitive and can be used without documentation. The help documents are easy to follow and relate to specific component.

5.4.3 Observations and Comments

Evaluators were asked to record their observations and comments in the log form (Appendix F), which was reviewed and served as a form of détente between the formative and heuristic evaluation. The following observations and comments were made.

i. *User Interfaces.* The desktop metaphor used for the GUI is helpful, but should support different levels of users, with different skills. Different levels of users will require different levels of information and use different components. The result of the heuristic evaluation showed that the GUI was fairly intuitive.

ii. *Development/Learning Environment.* The advantage of a fixed window layout in the development/learning environment is that it ensures that appropriate components are displayed, and allows an enhanced intra-screen navigation management. It allows text and graphics, applets, and other tools to be placed in pre-defined areas of the screen. Due to the modular nature of the GeNisa, different tools (viewer, toolbar, or dialogue box) can be displayed on the screen at the same time without overlapping. Novice users may have problems with basic screen management such as inadvertently moving windows off the screen, inability to restore a minimised window, etc (Nielsen, 1994), (Shneiderman, 1998). Therefore, a fixed window layout is the preferred method for displaying information in development/learning environment, where users tend to be novice users and where a sequence of events are to be followed. Although multiple windows may be preferable when used with predefined procedures, such as in Wizards, this should be used with care and attention.

iii. The system should provide some form of *Focus and Context* (Stuart et al., 1998), which may enhance interactivity. The user interface components and instructional strategies should guide the users through a specific instructional goal.

iv. *Use of the Toolbar* enables ease of navigation by providing short cuts to the underlying components. This distinction is necessary so that users are not confused with an option they would not require.

v. *Screen Management.* Screen management provides the ability to navigate through the different windows. This approach allows selective display and control of windows for displaying specific information and for manipulating information. It also gives the user control over screen sizes and allows his/her to navigate through different instructional components.

5.4.4 Heuristic Evaluation: Concluding Remarks

Heuristic evaluation provides supplementary information to support the questionnaire evaluation conducted in the preceding section. The questionnaire evaluation did not find any major problems, and also showed the use of the direct manipulation user interface to be an effective method of enhancing interactivity. The following conclusions were drawn from the heuristic evaluation:

i. The usability inspection method is a quick and cost-effective solution for conducting usability evaluation (Nielsen, 1994).
ii. The lessons learnt from the evaluation methods can be applied to the proposed methodology (Chapter 6), both in the user interface design and in component development.
iii. The traditional fixed-page screen format, as the only method of displaying the information, may be too restrictive for ITS user interface because users have different levels of needs and interactivity requirements. Also, dialogues, toolbars, scroll-windows should be visible on the screen, and should be consistent across applications and platforms.
iv. A rigid windows framework can be used to support novice users, but it allows more experienced users to by-pass the windows management system and may be beneficial in providing a more flexible development and instructional environment. Also, the developer should also be able to predefine windows, and set the windows accordingly.
v. A configurable user interface should take account of different user groups. Different tool bars can be used to support the different user groups with different tasks. Certain tools should be associated with the certain applications and should be designed to reflect the different users' requirements.
vi. Use as many forcing functions as possible. A forcing function (Normal, 1988) prevents a user from performing actions that are unwanted in a given context.

Nielsen's usability heuristic provides an approach to which GeNisa usability and systems functionalities is measured (Nielsen, 1994). The guidelines helped identify possible usability of the user interface and the effectiveness of the other components, although some of the conclusions drawn from the heuristic evaluation are similar to the conclusions drawn from the formative evaluation. Based on these evaluation results, it is feasible to infer that instead of providing the users (developments and learners) with a predefined problem-solving environment, the generic architecture should provide a flexible work environment for users to manipulate tools interactively.

5.5 Critique and Synthesis of Evaluation Results

Most of the components of the generic architecture were evaluated by using formative (questionnaire) and heuristic evaluation. The rationale for using formative evaluation is to examine GeNisa by using different user groups, which helped examine the components of GeNisa based on set of evaluation criteria. However, heuristic evaluation was used to complement the

formative evaluation, and it helped in identifying many usability problems that were not identified by the former.

The most important concern of this study was that there was no proposal for how to structure the evaluation criteria, in order to facilitate selection of range of component characteristics. Nevertheless, it could be said that the number of samples used for this study is limited, but. according to Monk et al., (1993) even three or four in a sample may be adequate. Also, besides the limitations inherent in a direct manipulation interface (Cohen, 1992), the generic architecture use of direct manipulation device (i.e. mouse) may limit interactivity on different platform and on different domains.

The design and implementation approach followed the use of object-orientated (OO) software engineering methods and synthesising knowledge-based methods for engineering reusable, coherent and maintainable components (Booch, 1994), (Sommerville, 1996).

The generic architecture uses a class library and a components-oriented approach to foster components reuse, design and sharing (Eriksson et al., 1996), (Musen et al., 1995). This approach may allow components to be used and developed in a uniform and centralised method. Moreover, components can be used as COM objects, which promotes platform independence, encapsulation and reusability (Microsoft, 1998). For example, a component functionality can be contained in COM wrappers (an operation that encapsulates a call to other library routine) and reused with different application and across platforms (Microsoft, 1998), (Rumbaugh et al. 1991). Each component uniformity is centred on its class definition, attribute relations and cardinality of reuse (Booch, 1994). This approach provides the following advantages: (i) components may use uniform events communication processes, (ii) when a component module is changed, the interfaces to other components do not have to be modified (Booch, 1994), (iii) components modules may be easily modified to use new types of knowledge or event processes. Nevertheless, this approach may eliminate duplication of effort required during development by: (i) promoting inter-component consistency preservation, especially across component boundaries, (ii) conceptual component schema may be simplified, (iii) cross platform utilisation of component functionalities and event processes, (iv) and unrestricted enhancement to components as required. These may promote implementation autonomy on multiple platforms.

If component reuse and sharing is to be fostered, components must use the same conceptual schema and maintain an appropriate view mechanism (Booch, 1994). However, the mechanism must allow for viewing different schema definitions and component attributes. ITS development

is often fragmented and less successful because of the different software tools and development methodologies involved (Murray, 1999), (Self, 1999). Therefore, GeNisa architecture provides numerous new capabilities that interoperability and reusability can bring to ITS research and development. Some of these capabilities include allowing ITS systems to take advantage of compatible commercial products, improving the cost-effectiveness and reusability of the components of ITS in diverse domains, and increasing the level of interaction between related ITS. These capabilities can lead to important benefits, such as increased efficiency of the development of courseware for different domains and using sets of teaching strategies. Reusability in GeNisa is supported by the modular design of its components and implementation method. The approach may allow not only reusability, but also shareability and portability.

Since most of the components are not tied to any domain-specific content area, this approach allows each component and its class library implementation to be easily customised. Component development could also be facilitated by the use of automated design and a class library definition interface. For example, an ITS component could easily be developed by importing existing class representation modules with their respective implementation methods.

The evaluation results suggested that class library approach may contain components that are too specific and vague and which lack component version management, i.e. components required for managing different versions of components. Also, components of the architecture may share the same events processes. Therefore, the components use facilities provided by the operating system, such as shared memory, message queues etc. for their inter-application process communication. The disadvantage is that all components must be executed on the same host, i.e. in the virtual machine (Gosling et al., 2000), (Arnold et al., 2000). Thereby, they have to share the host's resources such as virtual memory. The performance of each component will decrease as the number of concurrent component increases. Hence, the performance will deteriorate as more concurrent component processes are utilised. These issues form the basis for the enhancement proposals discussed in Chapter 6.

5.6 Summary

This chapter has discussed the evaluation of the generic architecture and has explored the various facilities that the GeNisa offers. Formative (questionnaire) and heuristic evaluation methods have been carried out, and the generic architecture has also been evaluated conceptually, by examining the system's component behaviour and functionalies. This framework has helped to examine the GeNisa prototype, as to how well it supports user activities, and to appraise usability, portability

and reusability. The evaluation has also examined the implementation of GeNisa, which has provided vital information that enabled this research to refine the development approach continuously and it has served as a form of détente between evaluation methods.

The results of the questionnaire dominantly confirm most of the generic architecture claims and demonstrate that users overwhelmingly found the GeNisa development and learning environment helpful, and that it provides appropriate tools for development and instructional use. Other interesting issues raised by the questionnaire included its suitability for courseware authoring, component implementation methods, unique features and functionality, module representation, reusability and portability, and general performance. Based on quantitative data obtained from evaluation results, participants indicated that GeNisa was reusable and portable. Also, the feedback from the heuristic evaluation indicated that GeNisa performance was good, the system is reusable, and evaluators found the system is flexible and that it supported their development activities.

From the qualitative data it is apparent that most users expressed positive attitudes toward GeNisa, by stating it was self-explanatory, interactive, reusable, and cost-effective. Also, the informal evaluation method seems to be applicable to ITS, although perhaps not ideal.

On the basis of the evaluation results, it seems feasible to infer that there are development benefits for developing ITS along the framework outlined in this research and may result in improved performance and reusability of ITS components.

CHAPTER 6

REFLECTION

6.1 Introduction

This chapter presents some practical solutions and theoretical approaches to address problems uncovered during this study and evaluation of the GeNisa (discussed in Chapter 5). The evaluation of GeNisa revealed a number of limitations in GeNisa. These particular limitations can be addressed by revising and extending the design and implementation concepts used in GeNisa. These refinements and improvement proposals have to be engineered so as not to introduce new difficulties to ITS development. This chapter proposes a theoretical approach and computational methods that address these limitations, and they are domain neutral and independent of applications. The proposals are drawn entirely from the experience gained during the design and implementation of the generic architecture and from reviewed literature.

The proposed refinements and improvement activities occur at two levels. Firstly, at a practical level, some elements of the GeNisa architecture are re-designed by exploiting additional features of the target platform used in the implementation of GeNisa. Secondly, at a more theoretical level, some proposals are put forward for ways to improve the theoretical tools available to help with building generic ITS components in general.

This chapter starts by reviewing the design and implementation processes described in Chapter 3 and 4, and the results of the evaluation presented in Chapter 5. This is followed by a detailed discussion of the proposed methodology for ITS development. Finally, based on the literature review and issues identified in the previous chapters, requirements for developing ITS and improvement proposals for ITS are discussed.

6.2 The Generic Architecture

This section highlights an area of the generic architecture that needs to be re-designed. The evaluation of GeNisa is confounded by several problems, which will be identified. Refinement

proposals to address these shortcomings will be provided. The refinement activities will support cross-platform utilisation as well as software modularity by using up-to-date technologies.

As GeNisa applications evolve, the components and their attributes may evolve due to (i) the need to support new component events, or problem solving methods; (ii) changes in domain knowledge/instructional contents; and (iii) the need to provide additional function or extensions to existing functionality. Furthermore, developmental activities are presented graphically to the user. However when building a large application with a small number of classes, the structures can become rather confusing. More so, different component modules have been implemented to separate component behaviour and their attributes so as to facilitate program compilation. However, experience of this research shows that component module compilation may limit abstractions for organising the program. The components-based approach as currently implemented in GeNisa could place demands on both the compilation and abstraction properties of the development language. Developing applications by using units of component modules requires defining a final compound unit that links the modules together. Therefore, in order to overcome these difficulties, the generic architecture should be re-designed so as to incorporate a mechanism that addresses these problems.

This reflective analysis requires re-design of some existing components in order to address these limitations. Therefore, refinement proposals are provided in order to improve the components functionalities in a more efficient manner. This section will identify fundamentally new ways of enhancing these component processes by leveraging current technology. The rest of this section reflects on some sections of the generic architecture that requires further refinement that uncovered during this study.

6.2.1 Generic Architecture for Intelligent Tutoring Systems

GeNisa has been designed and implemented on a PC to demonstrate the feasibility of a generic architecture for ITS, and to fulfil the this research objectives discussed in Chapter 1. The generic development framework is based on two perspectives: (i) development of sets of components and software models that are continuously enhanced during the development lifecycle; and (ii) abstracting the components into a class library. However, the following shortcomings are identified from retrospective analysis of the design and implementation methods and from evaluation results discussed in the previous chapter. This area of improvement revolves around

the components-based method and components class libraries. This includes the follwoing:

i. Developing a components class library is hard. For ITS to fully utilise a class library implementation method, it is necessary to understand a very wide range of problem characteristics and how to use component libraries to address these problem. This research has design each component class to be highly modularised so as to provide a solution for a specific problem or component behaviour. This requires the establishment of a prerequisite relationship between components in order to preserve component dependencies.
ii. Class library implementation of different components of the system has been used in order to foster reusability and portability (Booch, 1994). Therefore, a class library may contain a number of classes and nested modules. As different tools may have different composite schemas, which may be an event-dependent module, the class library approach may causes problems such as potential name clashes and interface structure.
iii. Inheritance relationships between components modules may, for instance, be public, protected or private (Booch, 1994). Therefore, the inheritance relationship between component classes imposes a very strong use relationship between classes. Hence, the export of the super class may result in the export of the subclass. Therefore, the inheritance relationship must be used very carefully, so that only the required super classes are exported.
iv. In general, component module instances can be partitioned into compound units, but these partitions must not overlap. Therefore, linking component modules are based on the library dependencies and linking hierarchy. This may increase the size of linking expression during development and the size of imported interfaces for an intermediate compound module. However, this approach may propagate the library dependencies, which may result in inconsistency in the software module.
v. The component libraries are based on a specific framework, for representation and for the kinds of component behaviour that can be initiated or stored in the library. Therefore, the component behaviour could be difficult to tailor for a new application.

Some of these issues will be addressed in the proposed methodology discussed in Section 6.3.

6.2.2 Reusable Components for Intelligent Tutoring System

During the design and implementation of the generic architecture, there was a clear emphasis on for the logical separation of the component content and behaviour from the main body of the architecture. This is because tightly coupled architecture may not be suitable for heterogeneous

utilisation, which may render reuse difficult. The GeNisa architecture framework allows the modification to be made to components without disruption of the operation, design and structure of the architecture (Booch, 1994), (Sommerville, 1996). The separation of the event processes from the rest of the architecture is possible because of the use of a separate event communication subsystem. This approach allows the events to be independent of the rest of the system. The architecture supports autonomy between the components such that different tools can be built from different sources. However, during the implementation of GeNisa, component-based approach was found not to adequately provide diagnostic reasoning capabilities. Also, because component interactions are bound towards an implementation, it may be difficult to reuse high-level design prescriptions for other applications. The component composition when used to develop applications may result in poor performances (Biggerstaff and Richter, 1987). Furthermore, some component functionality may not be required for new application or needs to be adapted to match the requirements of the application. This may be attributed to structural and semantic differences in the diverse sources of the component methods. Therefore, component evolution should be an integral part of components' reusability for ITS.

The flexibility of the generic architecture is partly realised through the events communication processes between the individual components (described in Chapters 3 and 4). The versatility of such an approach depends on the event-processing methods used during implementation (Booch, 1994), (Gamma et al., 1995). For example, if the event-processing system implements an inefficient protocol, the effect will be transferred to tools that use the same protocol. Another disadvantage of this approach is the lack of appropriate component version management as stated in the previous chapter. Furthermore, a component functionality may not be applicable in a specific situation as it is currently implemented. Therefore, it may need to be adapted, which can be done either by modifying the component's source code or by using wrapper (Box, 1998). Furthermore, both approaches have inherent disadvantages that may affect reusability.

Although an object-oriented and component-based approach constitutes a means for design, reuse and sharing of data (Ritter and Koedinger, 1997), (Roschelle and Kaput, 1996), (Suthers and Jones, 1997), as some component functionality in the generic architecture evolves, this may result in semantic conflicts between different components. Furthermore, each component attribute may be characterised by a certain number of different implementation approaches, different software/operating system versions, and implementation platforms. These characteristics may lead to the following. First, the developed applications must be incrementally updateable and compatible. Updating is necessary because the model must be able to accommodate the steady stream of new incoming data. Incrementality and compatibility are needed, otherwise the system

will have to reprocess all previous data each time new data is received and therefore will be slow and costly to use. Second, all developed components must be traceable in all phases from design stages to implementation, and reasoning subsystems. The requirement for tractability may be difficult in a constantly evolving environment. Finally, the potential for inadequate current application events indicates that the system should be robust and reliable when an unpredictable event occurs.

Another area for improvement revolves around the World Wide Web (WWW). The WWW approach facilitates deploying instruction across platforms (Brusilovsky, 1998). However, The WWW framework has an inherent dependence on network reliability and performance, which renders maintaining network integrity and security risks a challenge. Furthermore, deploying an application on a heterogeneous environment with different platforms (e.g. (Gosling et al., 2000), requires balancing the load of all applications across the platforms. As currently implemented, GeNisa does not handle the dynamic setting of ITS on different platforms. Furthermore, the user's environment is constantly changing during its life cycle (Clancey, 1992), because of the changing needs of the users. Therefore ITS developers must deal intelligently with a steady stream of incoming information/data.

The generic architecture adhere to Components Object Model (COM) paradigm (Box, 1998), (Cox, 1996). COM classes have unique tags (*CLSID*, a 128-bit descriptor), which has been found difficult to use. However, some proprietary software does not support COM e.g. Java language from Sun (Gosling et al., 2000), (Arnold et al., 2000), which may affect portability. Furthermore, COM approach may be inhibited by the use of programming languages that make the task more arduous. Therefore, additional flexibility can be gained through the use of proprietary mechanisms such as plug-ins, although, the disadvantage of plug-ins is that they are platform specific and must first be downloaded and installed prior to use. However, some proprietary software provides Application Programming Interface (API) to manage new protocols dynamically.

6.2.3 Knowledge Base

As currently implemented, component objects roles represent characteristics of components that may be independently referenced, or shared (Musen et al., 1995). This allows reuse of knowledge from different types of problem representation methods and organisation of the contents according to their inheritance characteristics. Also, the knowledge base has been developed so as

to support different courseware needs. As domain requirements change, knowledge can be added, or refined.

However, there are two factors that may hamper the effectiveness of the knowledge bases across domains and tasks: (i) the interaction problem (Bylander and Chandrasekaran, 1988) that is domain knowledge cannot be represented without knowing for what instructional task it will be used; and (ii) as currently implemented, the domain task provides prescriptions between content structure elements and instructional strategies. But domain task does not adequately account for different levels of instructional outcomes. Moreover, the tutorial remediation is largely driven by choice of tasks performed by the user. As currently implemented, tutorial remediations are chosen by the pedagogical agent. Therefore, a poorly chosen set of tasks may yield less effective tutorial delivery. This may result in poor interaction between the learner and the instructional system, which could result in a poor approximation of learner's knowledge (Salomon and Perkins, 1998). In addition, Sweller (1989) asserted that poorly supported problem solving activities might force the learner to rely on weak methods for problem solving. This implies that provision of appropriate remediation and tutorial structure makes learning activities more efficient (Vygotsky, 1978).

6.2.4 Automated Knowledge Acquisition Module

As currently implemented the GeNisa architecture uses knowledge acquisition process, which consists of the development of a domain knowledge level model of the expertise required for solving the domain problem. The knowledge acquisition module works by combining statistical inferences (Bayesian network) and database systems to acquire knowledge from both stored data and the user's interaction during instruction. During the development, it was decided to employ algorithms to extract knowledge according to the criteria presented in Appendix B. This in turn limited the number of domain behaviour that has to be instantiated. Therefore, content knowledge may be "shallow", which may require additional knowledge in order to support instruction. The prescription for domain hierarchical tasks strategies may be superficial.

This shortcoming raises the question whether it is possible to increase the reuse scope of some components of the generic architecture across a different domain. The rest of this chapter will address some of these issues and propose a methodology that is drawn entirely from the experience gained during the design and implementation of GeNisa and from current ITS literature.

6.3 A Methodology for Intelligent Tutoring System Development

Based on the experience gained during this research and on evidence obtained from literature (e.g. (Murray, 1999), (Bell, 1998), (Shute and Regian, 1990), (Koedinger et al., 1995), (Vanlehn, 1996b)), this section proposes a methodology for ITS development that seeks to address some of the shortcomings identified in the preceding sections and issues that were identified during the evaluation of GeNisa. The proposed methodology does not suggest re-implementation of the system, but it suggests some conceptual design and implementation strategies that could serve as guidelines for developing ITS, and is not committed to a single domain/platform. Furthermore, the proposed methodology provides logically coherent design processes that could serve as a reference "tool" in other to resolve design difficulties that characterises early stages of ITS development.

The proposed methodology seeks to address issues raised by using object-oriented and component-based approach for implementation of GeNisa architecture. The refinement proposals herein are structured with respect to ITS development tasks, in order to satisfy requirements on efficiency and improve interaction.

The methodology focuses on integration of independently developed components schema by enhancing developed classes with abstract data types to help in understanding and integrating the event-communication parts of different classes. This methodology provides an approach by which ITS can be developed from sets of existing schema classes and integration of existing ones. This approach is also concerned with reusing and sharing already developed schema (Musen at al., 1995) and in finding commonalities between the classes to be integrated. This section starts with a brief overview of the concept used to develop the methodology, and is followed by a proposed methodology for ITS development.

6.3.1 The Concept

There are two major schools of thought regarding development methodologies: (i) the scientific method, as represented by Dixon (1987), and (ii) the engineering method, as represented by Koen (1985). In the former, prescriptive methodologies are only developed following the development of accurate descriptive methodologies that lead to testable theories of a development. The latter prescribes that the prescriptive methodology be put forth based on the best available information, and then modified as necessary.

This proposed methodology is based on software engineering principles rather than in an analytic approach because engineering principles involves reuse of already existing solutions (Sommerville, 1996). Furthermore, engineering methods may support development and transition into active use of tools and techniques needed across platforms and domains. The objectives are: to identify the strengths and weaknesses of the design and implementation of GeNisa; to propose improved software components which may address the issues identified during evaluation; to develop components, techniques, and systems that can be used across platforms/domains. This approach might provide a useful methodology for considering strategies and mechanisms that are most beneficial from the perspectives of software engineering in general and reuse in particular.

Methodology in general "refers not only to (research) techniques or to inferential procedures, but also to the epistemological reasons for their choice" (Gebhardt, 1978). It can be inferred that methodology is a "*prescriptive*" procedure that allows research to be justified by recourse to identifying the strengths and the weaknesses of the development processes involved. Methodology provides "a very useful distinction between what is to be done next, who is to do it, and how" (Avgerou and Cornford, 1998). The methodology developed in this research consists of a structured approach with a clear and logical progression. It embraces both the independence and unified ITS development on heterogeneous platforms. The proposed methodology is independent of the development paradigm, and comprises a strategy for developing ITS from analysis of the domain and its contents to obtain the logical abstraction that represents different schema classes.

There are many advantages of this approach, the most important of which is that developers already familiar with development tools can leverage development time and skills requirements. This is possible because many of the features in proposed methodology fit well into existing frameworks, and developers can reuse knowledge in new domains and share encoded knowledge across different environments. Facilitation of reuse of components was identified as a fundamental requirement during the evaluation of GeNisa.

There are a wide variety of approaches to knowledge representation (Musen et al., 1995), (Musen, 1993), and knowledge that is expressed in one formalism cannot directly be incorporated into another formalism. Thus, in many cases, sharing and reusing knowledge will involve *translating* from one representation to another. As currently implemented, the only way to do this translation is by manually transforming knowledge from one representation to another (Musen et al., 1995). As stated earlier, the proposed methodology draws on the generic architecture framework and relates to the findings discussed in Chapter 5. The proposed methodology consists mainly of five

steps that may help ITS developers collect, analyse, and present information, in a highly interactive and iterative way.

6.3.2 The Proposed Methodology for Developing Intelligent Tutoring Systems

The main advantage of the generic architecture framework is its tenets for reusability and portability. Implicit in this approach is interactivity, shareability and modularity issues that could affect the functionality of a component. A methodology for ITS development, with stages representing steps for the natural and intuitive authoring process, is postulated. The main virtue of the proposed methodology for ITS is that it could serve as a theoretical foundation and a method that provides procedures for developing ITS.

Essentially, the proposed methodology consists of five steps: knowledge modelling, design of knowledge support tools, knowledge activities development, evaluation, and implementation phase. Figure 6.1. illustrates the major steps of the approach. Effectively, these phases represent how components classes may be structured during ITS development.

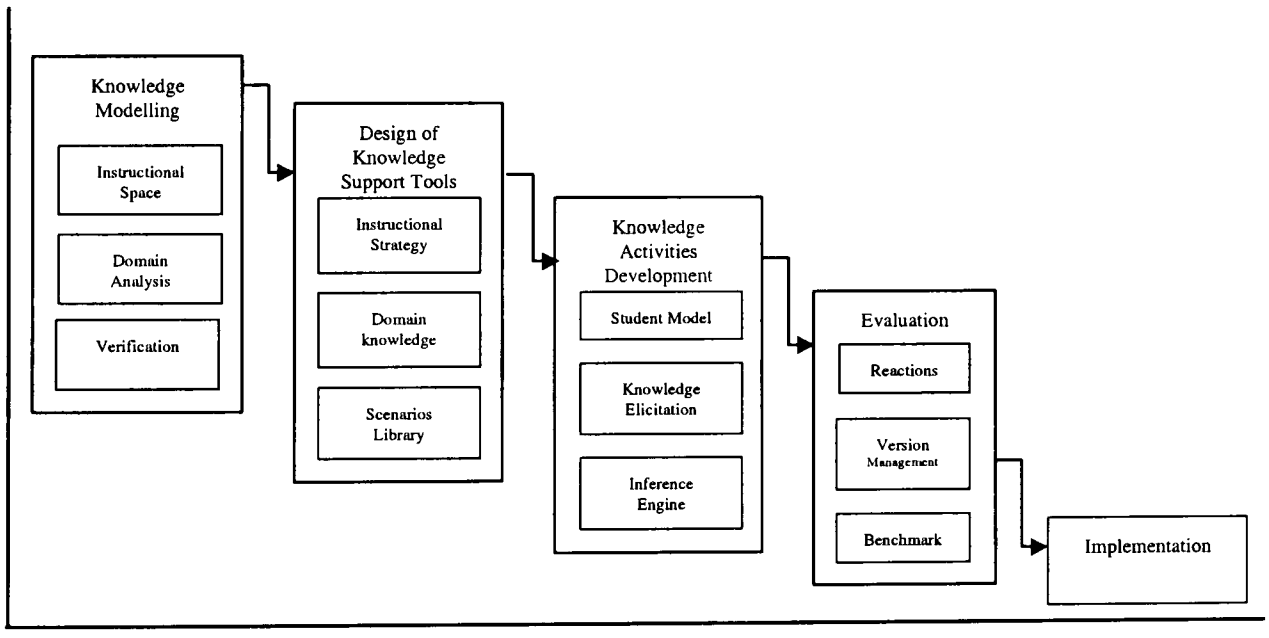


Figure 6.1 Schematic Representation of the Proposed Methodology

The output of each phase consists of a series of fully functional models of different ITS components. A more detailed description of this methodology is depicted in Figure 6.2. Each step divides the development processes into manageable phases and allows integration and extension of different classes. This may help to reduce the complexity of ITS development, by allowing different levels of abstraction about the systems' components. More so, this may increase the independence of the system's components, permitting component events and platform requirements to be localised as required.

These steps are depicted in a spiral-like model, which provides an efficient approach for transformation of development activities between different steps. The horizontal arrows extending from each box represent the expected outcomes of that step. Each step encapsulates operational activities, such as schema modifications, and component enhancement. The following sections provide a detailed description of steps defined for the proposed methodology.

Step I: Knowledge Modelling

This step involves preliminary operations to assist the developer to identify and formulate the general domain knowledge and define the semantics of the knowledge bases in relation to their domain (Chandrasekaran et al., 1992). Knowledge modelling involves modelling all aspects of system at different "levels" of abstraction. It involves a "systematic elicitation" of objects and their classes and the conceptual structure of the domain. It seeks to establish the core requirements for the system and to provide a model of the component's behaviour. This stage is open and its implementation is neutral. It may help a developer to delineate the domain specific components from existing/different sources. The knowledge modelling may be used as a guideline for decomposing complex tasks into subtasks. Furthermore, it can be used to provide flexible reasoning and problem solving methods (Benjamins, 1995), (Breuker and Van de Velde, 1994), (Chandrasekaran et al., 1992). The following issues are identified and addressed at this stage: components of the system, events and their distinct behaviour, problem solving methods and adaptability with regards to domain and possible changes as the result of user's actions.

This stage involves analysis of domain requirements and identification of component functionalities and events. This stage primarily involves the transformation of the domain tasks into hierarchical classification and lays emphasis on the events underlying the behaviour of the components.

An essential consideration at this stage is the analysis and design of the domain expert, discourse strategies, student model, and communication process. The knowledge-modelling phase helps to identify the major facets of the development and the sequence of activities and components events, with their degree of abstraction. The degree of abstraction may help to establish the component boundary and communication protocol.

All functional and non-functional requirements must be identified and specified at this phase. In general, knowledge modelling involves conceptualising and formalising the instructional "space". As currently implemented, GeNisa instructional space representation may not be optimal.

Therefore, by defining functional and non-functional requirements at this phase, an objects schema could be reused, which may reduce the development time and help to maintain consistency.

The educational, instructional strategies and pedagogical requirements constitute the instructional space. The instructional space involves the cognitive analysis, interactivity, structure and organisation of instruction components. It involves the production of a cognitively guided instruction, and dialogue that would bring about an improved learning with enhanced interactivity. The instructional space can be used to articulate an abstract tutorial structure, and ensure that domain knowledge is not limited to procedures for executing a tutorial task, but includes other instructional aids and case scenarios.

The instructional space is the user's first contact with an ITS, and involves different activities such as user-interface design, knowledge-elicitation methods, instructional strategies and navigational strategies. These are characterised by the following:

i. *Openness.* Instructional component may need to integrate information from different component repositories. This should allow developer to be able to define and customise components and view their schemas in a consistent way. Data and instructional events should be defined in a platform that is independent and language neutral, and is running efficiently.
ii. *Uniform didactic style* between different modules in order to provide a potential guidance for students, which may support the navigation over and through the knowledge corpus.
iii. *Support diverse domain and pedagogy development* and pedagogy by ordering of domain tasks into a tutorial sequence.
iv. *Extendibility.* Allows the developer to extend and customise the domain knowledge.

These characteristics partly establish the contents of the tutorial and discourse module. Once a target domain is identified, the instructional-space analysis is conducted based on these characteristics. The knowledge model is then transformed into domain-specific semantics. These semantics should capture the relationships between the relevant elements and describe how the elements interact.

Step II: Design of Knowledge Support Tools

The design of knowledge support tools involves the establishment of a non-dependency and unified representation of the data obtained from Step I. Step I helps to produce sets of models that

can be represented as domain component schemas. The design of knowledge support tools allows the developer to specify different component models, or create new ones by merging new/existing components schemas. It helps to establish mappings and maintain the consistency of the new and existing objects and the characteristics of the target platform for executing the system are made explicit.

The object-oriented paradigm is believed to reflect naturally the behaviour and structure of complex applications (Geller et al., 1991). It allows object schemas that consist of a collection of classes to be organised into hierarchies with their abstractions of the domain represented. All objects encapsulate data and reflect the dynamic nature of current domain objects, which may evolve over time.

A component class describes abstractions over common behaviour and structure between attributes of similar classes (Runbaugh at al., 1991). Therefore, component classes may be classified into categories in accordance to their functionality and content. This will allow the developer to identify common component functionality and to specify a common structure and behaviour in one component and reuse it for similar components.

The object-oriented paradigm can be used to structure the overall knowledge-modelling phase into component classes, and to define inheritance relationships, plus to reuse properties in subclasses. Each component class and their inheritance relationships may help to propagate structure and can be integrated together during development. Also, it automatically inherits the characteristics of this super class (Booch, 1994). This approach allows the developer to share commands and events processes between different, but similar classes types, and thereby foster reuse. This framework allows multiple versions of the component classes and attributes to be utilised during development. Furthermore, this will require a shared database to be maintained and supports Object Linking and Embedding (OLE) (Box, 1998), (Cox, 1996).

Components class integration requires the integration of the classes' behaviour and attributes. Component classes may be classified according to their schema entities (Ostertag et al., 1992), and integrated by using the predefined set of attributes for each class. This approach requires each class to be classified according to their problem-solving methods (Musen et al., 1995) before they are integrated. It is generally advantageous to perform this classification during the knowledge modelling phase and not retrospectively (Hoydalsvik et al., 1991). A developer may initiate component development process by identifying and selecting a domain classes from the problem class and instantiating them. For example the solution class allows the developer to define

solution and instances of the classes. This may involve a set of high level decisions on how to solve a problem.

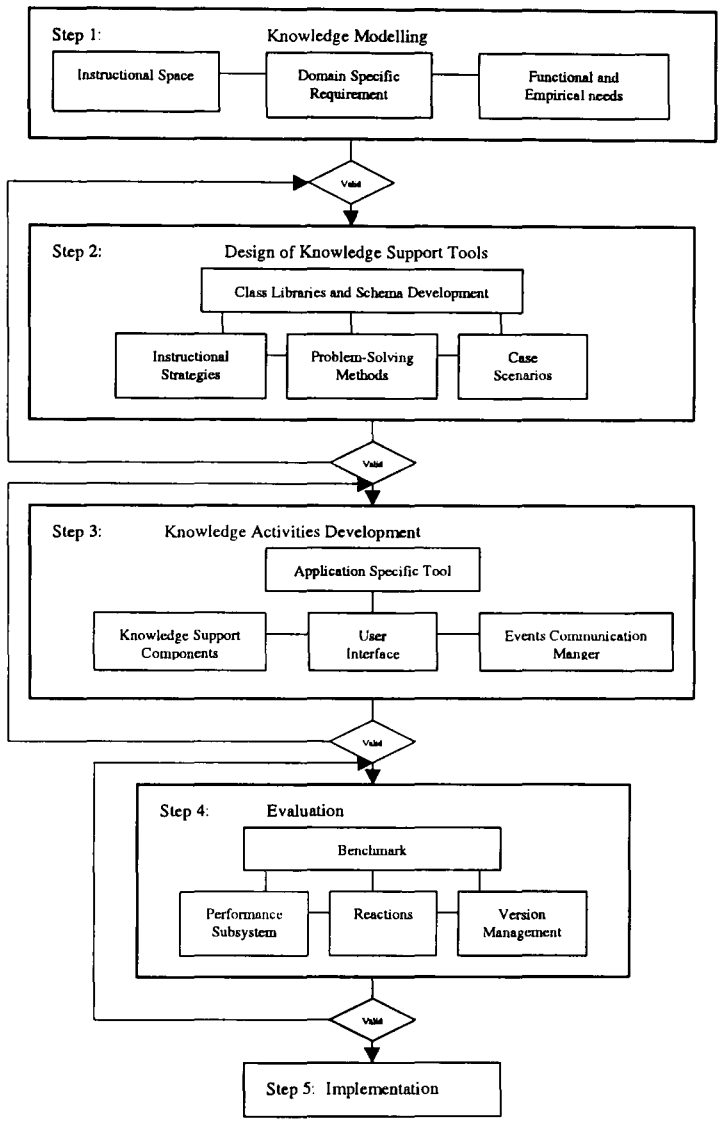


Figure 6.2 Proposed Methodology for ITS Development

This phase involves explicit representation of the domain knowledge, problem-solving methods (Musen et al., 1995), and case scenario into their representative component classes. This approach provides a basic building block, which developers can build on and provides powerful capabilities to express semantic dependencies between any different range of components. Furthermore, the representation formalisms may allow the developer to compare and validate independently developed components classes, problem-solving methods, and detect any conflicts in between different classes. A component might be defined to inherit (through inheritance links) behaviour from several other components. This research represented the component hierarchies as a directed acyclic graphs (DAG) (Pearl, 1988a), (Pearl, 1993), (Friedman et al., 1997), where the vertex corresponds to components and the edges correspond to inheritance links. This research assumes

that a component will behave in the same manner if it inherits all the properties of the parent component.

Therefore, a component propagation link is a direct connection between the two component classes. A class existing in one component can be propagated to all other components. The resulting class attributes will be a set of possible overlapping between different components attributes. The problem with this approach is that two or more components attributes may semantically overlap. This may be addressed by introducing real-world entity objects (Kurt, 1992). Therefore, a component integration is based on the relationships between the components objects (Sheth et al., 1988), their contents, and behaviour.

The design of knowledge support tools in this framework consists of connecting two directed acyclic graphs (Pearl, 1988a). New classes with different functionalities may be produced as a result of this connection. The main construct for allowing this is to connect component and application class attributes. Figure 6.3 depicts a DAG for abstracting different user-defined components into a Java code.

As domain requirements evolve over time, component contents and behaviour must be incrementally updateable in order to maintain efficiency. This ability to modify the components behaviour and contents "on-the-fly" provides developers with the option of making data structure changes to meet their changing requirements. The schematic diagram illustrates a method for abstracting a domain-specific application. The developer selects a component that needs modification. Based on the component contents attributes, the component may be abstracted into a new class by using a Bayesian network. Domain - specific attributes are applied or modified during abstraction to improve performance.

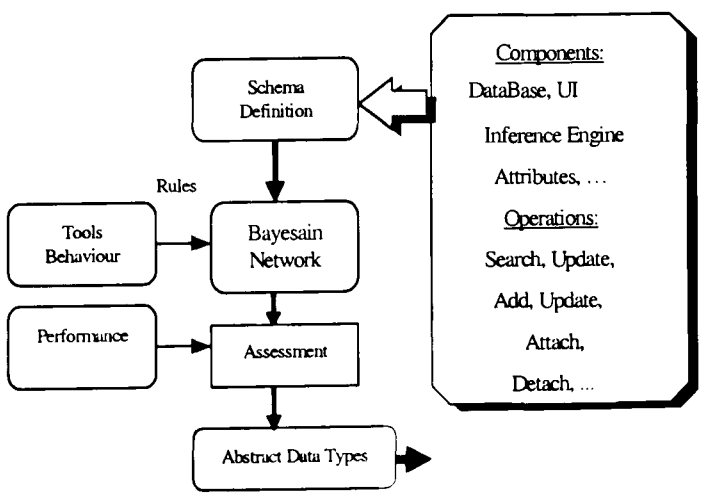


Figure 6. 3 Abstraction Method for a Domain-Specific Application

Step III: Knowledge Activities Development

This stage is concerned with the further development of different component classes into a functional component, with unique behaviour and contents. Steps 1 and 2 provide the input for deriving the component content and behaviour to be complemented as the representation of the domain features and required functionality.

As component class may have multiple views, views instances may intersect with different classes. Component instances become members of a view instance when created in the current context or by explicitly "joining" an existing object with the view instance. Each object must be created in the context of one view instance.

This phase involves component validation, as a component attribute and behaviour may have been poorly defined and implemented, but still function correctly at the interface level. This validation process helps to identify sources of conflict from the component classes and may help the developer to detect and resolve inconsistencies.

Instructional events communication is conducted by a software subsystem to provide applications with a separate communication module for sending events to other ITS components and receiving responses or requests from there. The subsystem is concerned with co-ordinating concurrent activities that might violate the consistency of the current tasks. These activities share the same objects and, most importantly, share the same component library.

This approach allows several components event processes to be run at the same time and communicate with each other by exchanging event messages. This enables components communication to be defined at a much higher level of abstraction and allows the user message-routing processes to be embedded in the architecture. It also provides a dedicated and application specific subsystem for communicating between other ITS components. Each module has a reference to an object of class, which provides the means for all communications between ITS modules and their processes.

User interaction and events could be represented as messages. During the construction of a message that represents a service request, the parameters of the service request are obtained from the communication subsystem and stored in instance variables of the respective message components. Most of the components developed could provide a COM and/or DCOM

(Distributed Common Object Model) interface that allows them to be used on heterogeneous platforms and with other applications.

Step IV: Evaluation

The steps in the previous sections describe the design processes characterised by domain specification and instructional design needs. This is followed by construction of a code unit for each component module. The above framework promotes evolutionary programming in a way that allows new classes to be added, which refines the behaviour of existing classes, rather than having to modify the existing code. This is an important feature, which allows for polymorphic programming; that is, the same code may be used for objects of different classes (Booch, 1994). Dynamic binding is often combined with a redefinition of operations in subclasses; that is, a subclass provides a new implementation for an operation, which is defined in one of its superclasses.

The fundamental advantage of this methodology is that a single modelling paradigm (objects, classes and hierarchies) is applied throughout the process of steps (steps I–III), as discussed in preceding sections. This approach closes the semantic gap between ITS components and their representation formalism, and reflects on the component contents, structures and functionality. This stage allows further enhancement and changes to be implemented through successive refinement, and manages post-delivery evolution.

The evaluation phase allows the developer to check the functionalities of the systems' components and to: (i) ensure that optimum performance is met consistently; (ii) test the efficiency of the system; (iii) help to identify future enhancements, (iv) ensure that non-functional requirements, such as, extensibility, interactivity and usability are satisfied consistently. However, these are more general GeNisa attributes that are applicable to any ITS development. Evaluation of ITS can be measured on the following levels:

i. *Reaction*. Test the application of the domain knowledge and the use of the instructional strategies.
ii. *Learning*. Suitability for educational use?
iii. *Application*. To what extent does the domain knowledge apply to real-world problems?
iv. *Impact*. What impacts have any of the instructional strategies and components on its development and deployment?

v. *Suitability of the components*. Developers must select components and methods that are considered most applicable to the domain knowledge and the environment.

vi. *Quality Assurance*. What quality assurance measures are provided? This should provide answers to the above questions and the analysis of static semantics. Quality assurance processes must also help improve reuse capabilities by means of identifying obsolete variable declarations and the use of uninitialised variables.

vii. *BenchMark*. Development of a set of criteria functionalities that will allow benchmarked comparison of ITS components. Activity will involve comparison of the following: (i) effectiveness of ITS components against sets of other AI components (Murray, 1999), (Self, 1999), (Wenger, 1987); (ii) effectiveness of intersectional intervention and discourse strategies; (iii) interaction processes; (iv) cognitive load "balancing" with the user's task; (v) knowledge representation formalism; and (vi) dynamic load balancing on the network. Measuring execution times is based on operating system primitives can be given in different terms. For an ITS performance evaluation, the most important execution time is the elapsed "real-time" of an operation. This time will indicate how long a user interacting with a component will have to wait for the operation to complete before starting another operation.

Therefore, it is feasible to infer that ITS that are implemented on different platforms ((Murray, 1996); 1997), (Anderson et al., 1990), (i.e. operating systems and programming languages), may differ significantly with respect to the functionality provided. However, components developed by this methodology may overcome the brittleness inherent in conventional ITS development methods by reusing existing components and using baseline architecture for implementation (e.g. (Sparks et al., 1999), (Murray, 1998)).

Components developed must be tested against each of the standard criteria on a benchmark before implementation. This approach will ensure that ITS components are consistent with industry standards and provide a uniform approach to which performance can be tested. This ensures that consistent design of components will foster reuse and ease of learning.

The evaluation approach should ensure the system's effectiveness, aspects of user interactions and the deployment environment. As application environment may dynamically evolve, it should involve unanticipated change in the user's environment. This is particularly useful for delivering instruction over the Internet. Hardware and software considerations must include system support for inter-operability in a heterogeneous distributed environment (Blair, 1994).

Step V: Implementation

Implementation depends on application area and underlying design perspectives. However, the proposed methodology delineated in this Chapter is an implementation independent framework. As such, components can be implemented as objects or compositions of components and packaged as independent pieces of ITS tools. Furthermore, components should be capable of offering a range of services across a variety of platforms. Implementation level delineates architectural tools and communication between these tools, including software modules, such as inference engine, event managers, etc., and communication protocol between them.

The class library and the Component Object Module (COM) approaches described in this section may provide all ITS components functionality and hence support the provision of a uniform style of access to all applications. The developed components should be tested and different kinds of components produced, including a technical components (e.g. Application Program Interface (API) and a user manual). Implementing the proposed methodology in a strictly sequential order is less feasible (Boe, 1988). Instead, the tasks should be carried out simultaneously.

6.3.3 Usability of the Proposed Methodology

The methodological framework has to satisfy many functional and fun-functional requirements posed by different platform requirements and users. Many of the reviewed ITS development methods are too slow, not portable and not flexible (Murray, 1998; 1999). The methodology outlined in this section may provide cost-effective approach for developing ITS by integrating components in many forms. The proposed methodology offers guidelines for interoperability and reusability of ITS components, which is aided by shared ontology representation (Musen et al., 1995), (Ikeda and Mizoguchi, 1994). Shared ontologies help developers communicate the contents and functionalities of their components and implementation strategies (Ikeda and Mizoguchi, 1994). It also provides tools to facilitate the authoring process and minimise the effort and resources required for development. Furthermore, domain requirements may be stated in terms of needed functionality, and non-functional requirements such as extensibility, portability, or scalability. The proposed methodology provides developers with the means of making design choices based on knowledge of available software components and their functionalities. Also, the multiple levels of abstraction inherent in the proposed framework will allow developers to choose to reuse the design at one or more levels as required. The proposed methodology, if adopted, may provide the following benefits:

i. Improved maintenance of components, contents and behaviour, by allowing compatible applications to be integrated during development, thereby reducing duplication of effort, which could be costly. Replacing existing software by new software may also be expensive. Therefore, existing systems should be evolved (i.e. adapted and/or enhanced) to keep up with new and existing requirements.

ii. Components may function as COM objects (Box, 1998). COM objects allow application objects (or class libraries) to be transferred to different platforms without the need for programming and promote encapsulation and object reuse. GeNisa class libraries are COM-compliant, which provides platform-independence.

iii. Robustness against software and hardware failures, as components are represented as classes and as COM libraries. A components class library can be developed by different developers, and be independent of the domain contents, and provide a high degree of flexibility in development. This approach also allows components to be incrementally developed and integrated.

iv. Openness, and modular design for developing ITS components allows not only reusability, but also shareability and transportability (Booch, 1994) on different applications and knowledge to be used on heterogeneous platforms. ITS components may utilise different processes and data that are physically located in different places. It is important to strive towards *location transparency*, and present the user with a unified view of the system.

v. Increased reusability of the components of ITS in diverse domains, with increased levels of interaction between related modules. These capabilities can lead to increased efficiency of ITS development for different domains by using a set of domain-independent components.

vi. The framework allows ITS components to be structured into different independent, modular perspectives. Separation of components, as to content and behaviour from the rest of the architecture provides an enhanced-events processing mechanism between components by separating tutorial events from other activities. It allows the possibility of identifying components that are similar in structure and behaviour. The approach allows ITS components to be easily instantiated and reused. This approach strives to preserve the transparency of each class library, so the components of the system remain unaffected by changes in domain or hardware platforms.

vii. *Consistency and Standardisation.* The proposed framework promotes consistency and standardisation. ITS has consistently changed over the past years (Shute, 1994), and the issue of consistency and standardisation in design, development and deployment methodologies has become more paramount. Consistency and standardisation in ITS components functionality should cut across different platforms and domains. Consistency and standardisation should not involve a trade-off between non-AI and AI features. It is also

important to ensure that development standards are set and met consistently. Furthermore, all components will have standardised interfaces so that ITS tools are easily portable between different hardware and operating-system platforms.

viii. *Application Integration.* The application integration should allow the integration of third-party applications for instructional use. Application integration consists of management and editing facilities, knowledge acquisition, problem-solving methodologies and management tools embedded into an integrated environment. Also, integration almost involves an interactive process whereby attributes of existing components are tuned to allow new concepts to be integrated.

In addition to these levels of interoperability and reusability, the proposed methodology provides an evolvable environment for ITS development and deployment. Furthermore, the core of the proposed framework is the hierarchical software-engineering viewpoint, and the nature of separation of the design and implementation approach, which allows for class hierarchies to be evolved in different directions.

6.3.4 Trade-offs Between Traditional and Proposed Methodology

The development of ITS that combines aspects of AI and object-oriented techniques requires a methodology that provides easy techniques for manipulating each component class and attributes. This approach is based on the selection of a representation that is isomorphic to other classes. Cox (1996) argues that modular architectures are required for encapsulation and management of difficulties inherent in software. This viewpoint emphasises the need for a generic architecture that allows software to be flexibly adapted and extended to meet diverse needs.

The sets of trade-offs presented in Table 6.1 are drawn from experience gained during design and implementation of GeNisa and from reviewed object oriented and AI literature. The hierarchical nature of the methodology creates a gradual transition between the design processes and software development. It provides an explicit "road map" for separating the different design concerns that need to be addressed in different ITS components.

The first stage allows the developer to analyse the instructional activities and the tasks, problems and corresponding knowledge needed. Many of the trade-offs have emphasise component reuse, portability and interactivity, which satisfy some of the objectives of this research. This stage also identifies low-level representations that are difficult to specify during the early phases of ITS development.

The trade-off of the proposed methodology (shown in Table 6.1.) contrasts attributes such as representation approaches, analysis methods, ITS components, interactivity, reusability, portability, and representation methods and techniques. Also, each of the attributes is to bridge the void between the features offered by traditional ITS and those required by multi-platform applications. Furthermore, the hard coded approach inherent in traditional ITS development may limits the reusability of the components and introduces implicit assumptions into the system architecture that make it difficult to combine components behaviour in a new application.

Table 6.1 summarises the trade-offs between the ITS development methods, conventional approaches and proposed methodology. The table shows that the proposed approach supports incremental ITS development and reuse, making it suitable for building different components, and provides dynamic interaction between different modules. This allows developers to leverage existing tools or classes easily without the need to change the source code, thereby enabling easy extension to the tool or the ITS that uses the same architecture.

The hierarchical nature of the proposed methodology may allow ITS developers to become more flexible in their development by identifing an alternative approach for component development and code reuse ability. Moreover, it recognises the similarities that exist in an ITS system's structure and behaviour. While making the distinctions between various ITS development methods, the complexity of ITS development can be reduced. Similarly, it increases the ability of the developer to analyse and manipulate the system's components at different stages of development.

In the context of component class integration, it has been recognised that separating the structural and semantic dimensions of classes allows integrating classes that are structurally similar, but semantically different (Gellar et al., 1991). Classes are considered semantically similar if they model the same objects in the application domain (Gellar et al., 1991).

The methodology supports a coherent approach to ITS development and flexible implementation strategies. The approach is different from a conventional approach that allows the developer to reuse and modify existing component attributes, in a highly interactive and iterative way. Other main advantages are summarised as follows:

Table 6. 1 Summary of the Trade-offs between Traditional and Proposed Methodology

ITS Development Methods/ Features	Conventional	Proposed Methodology
Flexibility with different objects	Objects as well as relationships can be inflexible	Highly flexible with different objects
Knowledge consistency	Partial	Consistent
Knowledge representation	Frame/node level	Whole object and attributes
Inference mechanism	Rule-based	Quantitative
Authoring environment	Domain specific	Integrated environment
Domain knowledge	Deterministic, structured, goal based	Task-based, free exploration, evolvable
Tutorial discourse	Consistent	Customisable and consistent
Schema evolution	Partial, relational	Schema reconstruction
Development language	Procedural, e.g. Lisp	Object-oriented, 4GL
Instruction	Abstract, reactive	Cognitively guided, interactive
Knowledge base	Domain-specific, semantic network	Reusable knowledge bases, object-oriented
Platform	Single platform	Multi-platform support.
Reasoning methods	Bug library, overlay	Cognitively guided responses.
Development time	Comparably high	Relatively small, leveraging existing objects and writing code only as necessary.
Knowledge acquisition	Manual	Automated
Modularity	Minimum	High
Inter application events	Pre-defined	Mixed initiative, dynamic
Reuse	Low	High
Enhancement potentials	Costly and timely	Can be extended with minimum cost
Semantic cohesion	Low due to development methodology	Maximised during development
Version management	Minimum	High

i. The knowledge about the system behaviour is general and can be reused because all objects are developed from the same class library and may share the same event-processing protocol.
ii. The methodology can easily be extended, i.e. the baseline architecture requires less programming and domain analysis effort. This may also reduce the development costs.
iii. The reasoning mechanism functions as a dynamic diagnostic agent during development and continuously checks the design for potential errors and incomplete solutions.

By using AI and object-oriented techniques, the abstraction of each component class can easily be achieved by using the class definition and instances of the objects. Inheritance of the class attributes is fostered by the hierarchical class representation. These properties make the proposed methodology a highly viable development paradigm in various ITS domains.

6.3.5 Evaluation of the Proposed Methodology

Evaluation of the proposed methodology cannot be done without considering other external factors such as the developers' experience, the characteristics of the problem space (Monarchi and Puhr, 1992) and the broad spectrum of other technical and non-technical requirements.

Some components of the proposed framework vary widely in purpose, and use. However, this evaluation compares the various options for implementing some of these components. For example, a typical ITS may consists of the following components: user interface, database, inference engine, and discourse strategies. The user interface is often a combination of buttons, menus, and toolbars. Therefore, using the proposed framework may help produce a more usable and consistently better user interface.

Table 6.2 evaluates the benefits for implementing the proposed methodology by comparing features of the generic architecture presented in Chapter 3, the implementation approach (described in Chapter 4) and the conclusions in Chapter 5 with the Standard User Interface Guideline (Nelson, 1993) and design concepts of ITS development (Self, 1992). It is assumed that applications are developed on the basis of the steps outlined in the proposed methodology. Also, unless the design requirements of target application are extremely limited and specific, developmental activities in ITS cannot be considered without the combination of the different methods outlined in the proposed methodology.

Table 6. 2 Evaluation of the Proposed Methodology

Criteria	Remarks
Instructional Flexibility	Very flexible. Tutorial strategies are represented as probabilistic inference, which can be readily adapted.
User Interface Flexibility	Very flexible. Since the UI is developed in Java, which can easily be extended.
Data Volume	Through selection of an appropriate database, any data volume can be addressed.
Multi-User	Since the data is stored in a database, multi-user applications can be built.
Sharing Data with Other	The use of database, and class libraries (e.g. COM components) it is

Applications	readily available to other applications.
Installation/Distribution	Supports dynamic link libraries, COM objects and custom controls.
Portability	Portable. Multi-platform support, e.g. Windows and Unix environment
Developer Skills	Knowledge of Java and data connectivity, and OLE Must have good knowledge of the database being used.
Development Time	The development time for this approach may be typically comparable low.
View Management	Component objects are organised according to their view types.
Interactivity	Enhances interactivity, cognitively guided support functions
Object-oriented	Supports object-orientation, with enhanced interactivity and inheritance.

6.4 Improvement Proposals for Intelligent Tutoring Systems

This section addresses possible methods for enhancing ITS development. These improvement proposals are based on the experience gained during the design and implementation of the GeNisa as stated earlier. The improvement proposals are based on the features that have been implemented in GeNisa (described in Chapter 4) and ITS features that should be enhanced.

This research has identified the following requirements for ITS development that are largely elicited by a retrospective analysis of the development processes used in the research and requirements for generic architecture presented in Chapter 3, plus reviewed ITS literature. Some of the GeNisa architecture requirements were examined in order to identify specific effect on portability and reusability. Essentially, these improvement proposals must satisfy two general requirements: (i) it must easily be mapped onto the epistemological model in order to support knowledge acquisition (e.g. (VanLehn, 1996b), (Half, 1988)) and for defining associations between interrelated pieces of knowledge and (ii) it must allow the use of various knowledge representations and reasoning techniques. More specific requirements are:

i. *Reuse*. It should be possible to use existing components classes and import new ones, without any limitation to the size of the objects, decrease in functionality or to the maximum number of such objects that the system may contain, being imposed by the system. This approach will allow the system to be scaleable i.e. the ability of the system to manage effectively large numbers of users, components and class libraries (Booch, 1994).
ii. *Platforms*. The system should support multiple and distributed platforms. This approach will provide the ability to integrate application across platforms and support WWW.

iii. *Authoring*. ITS authoring environment must exhibit a flexible structure, accommodating objects of varying types and their relationships. Authoring environment should allow developers to create new objects with different semantics, and develop its refinement by allowing more complex functionality and providing wider choice for the user's interactivities.

iv. *User Interface*. There must be a domain-independent user interface for courseware deployment and development. This is necessary if the design representation is to be integrated with legacy systems, such as editing tools and application software.

v. *Version Management*. Components version management support is needed. This approach is based on automatically maintaining changes when component classes are modified, such that existing application program and their components are not affected.

These requirement proposals are in contrast of the requirements for the generic architecture (discussed in Chapter 3). These proposal may allow an ITS developer to satisfy as many application requirements as possible, which may be difficult to identify during the early stages of ITS development. Besides, the requirements for the generic architecture as manifested in the implementation of GeNisa, may be limited in the approach used to develop the system. These observations require examining the practical feasibility of the component-based method for developing ITS, the knowledge requirements for executing the behaviour of the components and some fundamental constraints in shaping the representation of the domain knowledge for use during instruction.

As a result, this research proposes an improvement to the shortcoming that are attributed to the design and implemented methodology used for this study. This research adopted an object model, which allows storage of different components, and can be structured into different views, thereby allowing the creation of relationships between them. This approach will encourage rapid application development through the reuse of components and their classes. Further areas for improving GeNisa architecture have been described in Chapter 5.4 and in Sections 6.4.1- 6.4.5. in this chapter.

6.4.1 Architecture Requirements

This subsection proposes features that could be enhanced in GeNisa and in ITS in general. It identifies a possible set of functional requirements that should be supported and provided by an ITS development environment in the various stages of its lifecycle.

Architectural requirements are based on factors that may affect the development processes and knowledge representation formalism. This stage requires the employment of the combination of different tools with different tasks. This includes:

i. *Schema Management.* This represents the ability to manage the contents of the knowledge bases. It includes support for modifying the contents and their classes in the knowledge bases, and integrating new ones. Schema management for ITS would allow dynamic changes of data, and supports merging new and old schemas. Schema management would provide an approach to add, remove and change attributes of the schema definition, automatically or dynamically at run-time.
ii. *Dynamics*. Dynamic requirement involves intelligent characteristics of the ITS modules and their events to change procedure and routines. The goal is to allow instructional events that dynamically adapt to users' behaviour and needs. It will allow the developer to plan a tutorial discourse and combine appropriate case-base scenarios, and dynamically adapt the tutorial goal.
iii. *Portability*. This specifies the need for developers to build applications and deploy them easily across the Internet or Intranets. This allows the architecture and the courseware to be accessed from anywhere, through a multi-platform environment. It should allow the use of a wide array of class libraries developed by other vendors. This will leverage tremendous extensibility to broaden the scope of the development. The fundamental requirement is the ability for the courseware to be developed and deployed on heterogeneous platforms.
iv. *Extensibility*. The schema consists of a collection of classes, organised into hierarchies, which represents abstractions of the domain. Objects are created as instances of classes, encapsulating data. An extensibility characteristic is the support for evolutionary programming so that existing programs may be extended with new classes without affecting other parts of the system.
v. *Knowledge Representation.* The knowledge representation formalism involves the use of common data formats, schema reconstruction, reusability, and allowing the data to be easily extended and modified. The knowledge base should represent an interrelated set of different instructional strategies, problem-solving methods and different case scenarios. Knowledge representation must be specified with multiple perspectives and should focus on global definition of objects and reuse of components.
vi. *Validation*. Integrated provisioning of functionalities for the validation and verification of the correctness of a component should be considered as a mandatory aspect to be supported. It could be conducted according to the domain and implementation environment. This should consider issues such as global requirements and resource constraints.

Knowledge representation language should support transparent data exchange and integration. This would allow the encapsulation of the components between heterogeneous data repositories and applications that need access these repositories. Besides, this may enable access to the knowledge bases from applications written in different programming languages. Also, the architecture requirement emphasises the logical separation of the event and application communication components from the main body of the architecture. This approach will allow modifications to be made to any parts of the system without disruption of the operation, or structure of the architecture.

6.4.2 Implementation Requirements

On the basis of experience gained in this study, it is reasonable to infer that the design and implementation of the generic architecture does not address matters of implementation efficiency. However, it is desirable to keep the step from design to implementation short, without sacrificing the component functionality and behaviour. This is essential in order to reduce the effort required to ensure consistency between components functionalities, during implementation.

Implementation requirements are cross-functional and specialised requirements that address current and emerging ITS implementation and platforms needs. These include:

i. *Interoperability.* This is to ensure interoperability between different ITS. Interoperability becomes more important in performing functions such as data transfer, knowledge base management, etc. Interoperability is essential if there are differences in platforms, operating environment, and knowledge representation formalism. The use of standard protocols will facilitate the interchange, addition, or reuse of components. Interoperability and reusability may be aided by shared schema definition and allow the export, import and use of any data format from different sources.
ii. *Robustness.* Developers should easily build applications that meet their domain activities, by combining ITS modules with their appropriate event-binding processes into a coherent method. This approach allows developers to quickly develop and refine current applications by assembling the components into the application.
iii. *Integrated Development Environment.* Consists of an authoring environment with an extensive library of multimedia classes, case-scenarios, images, animations, sounds, and application software. Other components includes: intelligent aids to help in the development of using objects, defining properties and behaviour, specifying user interaction, and setting up integration with the database, a visual, point-and-click facilities for class definition, and

object-editing facilities. All ITS functions, which should be integrated in a cohesive and graphical environment. This may significantly increase productivity. It would support the development of ITS without programming.

iv. *Object-Oriented Approach.* Object-oriented technology is particularly well suited to handling the complex data structures and large data volumes required for current ITS applications. This would provide the robust foundation needed for including hypermedia, and its integrity, security, transaction management, and performance. Implementation of the system should be platform-independent.

v. *Navigation.* The behaviour of an object, i.e. how it responds to user interaction, instructional events, database messages, and other events, should be defined in terms of a simple event-action metaphor, requiring no scripting.

vi. *Instructional Document Repository.* The ability to integrate databases of documents from different sources requires a mechanism for document integration, retrieving facilities and refinement processes. This would allow instructional documents to be stored in a set of heterogeneous databases. The schema of such database changes is often caused by changes to the domain knowledge, changes in the database structure, or in the operating environment and platforms. This also provides support for schema evolution in the databases so that their contents can be improved without having to change any tutorial contents, and representation formalism. It must be consistent in design and presentation regardless of platforms.

vii. *Incremental Development.* Reusing existing architecture could serve as an underlining platform for further development. This provides an experimental platform for testing various concepts and methodologies. It also allows the functional architecture to be enhanced to satisfy current requirements.

viii. *Data Models.* Multiple-object model perspective as opposed to single-object views. The data model should be configurable and extensible so that new objects and data model may be incorporated.

ix. *Components Integration.* ITS development environment must interface with other graphic packages, applications software and databases. Component integration provides the ability to automatically exchange data with other applications and across platforms. It must support encapsulation and integration of different packages.

x. *Location Transparency* allows applications to be developed without knowledge of the location of data. This requirement is necessary if the developer is working from different sites.

The choice of development language is also a vital consideration because some programming languages have inherent technical and conceptual constrains that may affect the development processes.

6.4.3 Security and Constrained Execution

Providing a secure environment for developers/users to access application on heterogeneous platforms and networks imposes some security implications. Only password security was implemented in GeNisa. Security violation can be prevented by:

i. *Algorithms.* Encryption algorithms, self-verification programs can be employed to ensure that application programs are not compromised.
ii. *Code Verification.* Applications should employ an authenticated code mechanism for verification and constrained execution of programs. This should provide functionality to perform a global syntax and type check of the specified components. This involves checking all the constructs and declarations. This could also involve a global check to detect errors.
iii. *Execution Restriction.* A small number of memory and duration is allowed during execution and the program should terminate automatically.

These implementation requirements provide a comprehensive and flexible method to guarantee the integrity of data and to monitor database operations across platforms. Furthermore, the improvement proposal involves every aspect of ITS development (including design and implementation, coding and documentation) and object oriented software engineering is implicit in the proposal.

The issue of generic architecture involves not just representation of a particular method but also epistemological questions. These involve instructional relationships between knowledge-based systems, ITS components and issues that relate to reusability and portability.

6.4.4 Dynamically Programmable Student Model

Most object models only allow for operations to be dynamically bound, whereas attributes are statically bound and are often only available internally to the objects operation.

However, each object in the student model may inherit features of a super-object if they are represented in an object hierarchy. The hierarchy should encapsulate the behaviour for the

classes, and each of the objects in the structure should have different roles. The object hierarchies may dynamically change by adding and removing objects to an existing hierarchy. This technique may be used to separates problem-solving methods, diagnostic and didactic decisions from the student's actions. This approach can facilitate the use of specific characteristic of the student model during implementation rather than hard-coding a reference to a particular implementation. Some existing objects may be programmed to extend a specific object hierarchy by extending a specific feature or attribute.

This approach in contrast to the GeNisa student model (discussed in Chapter 3 and 4) would allow the student model to be used as COM objects, and can be used across different. The ability to assign both static and dynamic variables actively into the student model will extend the functionality of the student model from a basic "perturbations" method to a dynamic model. Also, the exact functionality of the student model does not have to be determined during development.

6.4.5 Reasoning with Multiple Diagnostic Components

This addresses issues of how to deal with the simultaneous presence of multiple events during instruction and for content diagnosis. For instance, the student model might detect that the user is incorrectly applying an analogy, and simultaneously requesting an advanced course unit and provide a content scenario.

Consequently, if the two situations represent an ambiguous event, the reasoning system should propose an appropriate discourse. An adaptive heuristic can be developed to guide the user during instruction and should be incorporated into the student model. Reasoning with multiple diagnosis should also include the enhanced precision with context-sensitivity of user needs, and alteration of the reasoning process to avoid reoccurrence of a similar failure.

The proposed development methodology could integrate these tools and techniques to provide a comprehensive ITS development, validation, and maintenance environment. Each of the requirements can be influenced by different factors including operating platforms and environment. It may not be sufficient to combine a couple of these requirements in order to conduct an ITS research or development. ITS development processes have to be carried out in parallel and all development sequences must be carefully synchronised.

These improvement proposals were drawn from the literature review, requirements for generic architecture and the experience gained in developing GeNisa. Table 6.3 provides the summary

and comparison of the proposed improvement with conventional ITS. The table illustrates design issues, which characterise ITS development, and identifies the key elements that are most important for ITS development (Self, 1999), (Murray, 1999), (Clancey, 1992), (Breuker. 1990).

Table 6. 3 Summary of Proposals for Improvement of ITS

CRITERIA	CHARACTERISTICS	FEATURES TO BE INCLUDED/IMPROVED
Architecture	Domain and platform specific.	Factors that may affect the development processes and knowledge representation formalism.
Schema management	Series of models at different levels of abstraction, e.g. KADS.	Multiple perspectives and global definition of objects and reuse, and incrementally updating classes.
Dynamics	Epistemological structures of expert model (Wielinga and Breuker et al., 1990); exploring and modifying a repertoire of models (Linn et al., 1994).	Encompassing dynamic changes of data, and the ability to merge new and old schemas with domain knowledge.
Portability	Semantic networks/rule-based representation (e.g. SOPHIE, Brown et al., 1982).	Object-oriented class libraries, heuristic methods, and COM support.
Extensibility	Minimal - rule-based, e.g., GUIDON (Clancey, 1982)	Supporting evolutionary programming, existing component may be extended with new classes.
Validation	Minimal - e.g. PROTÉGÉ-II (Puerta et al., 1992), KADS (Breuker, 1994).	Integrated environment for the validation and verification component.
Development	Minimal - e.g. KADS (Breuker, 1994).	Cross-functional and specialised requirements and platform needs.
Interoperability	Minimal - e.g. PROTÉGÉ-II (Puerta et al., 1992), KADS (Breuker, 1994).	Interoperability and reusability is aided by shared schema definition and allow the export, import and use of any data format from different sources.
Robustness	Minimal - KADS, KIT (Linn et al., 1995).	Continuous refinement and extendibility with new requirements.
Integrated development environment	Toolkits implementation, e.g., PROTÉGÉ-II (Puerta et al., 1992), VITAL (Shadbolt et al., 1993), KREST (Steels, 1990) and KIT (Linn et al., 1995).	All development activities to be integrated in a cohesive, visual platform and to facilitate reuse of code and simplify maintenance. Support development without programming.
Navigation	Minimal e.g. PROTÉGÉ-II (Puerta, et al. 1992), KADS (Breuker, 1994) and KIT (Linn, et al. 1995).	User interaction and events should be defined in terms of a simple event-action metaphor, requiring no scripting.
Incremental development	Partial, e.g. KIT (Linn et al., 1995), KADS (Breuker, 1994).	It also allows the functional architecture to be enhanced to satisfy current requirements.
Data models	Partial, e.g. KADS (Breuker et al., 1994).	The data model should be configurable and extensible so that new objects and data model may be incorporated.
Components integration	Partial, e.g. KIT (Linn et al., 1995), KADS (Breuker, 1994).	Provides the ability to automatically exchange data with other applications and across platforms (location transparent and external interfaces data exchange).

Security and constrained execution	Partial, e.g. PROTÉGÉ-II (Puerta et al., 1992	Providing a secure environment to access application on heterogeneous platform and security implications.
Code verification	Minimal, e.g. PROTÉGÉ-II (Puerta et al., 1992)	Applications should employ authenticated code mechanism.
Dynamically programmable student model	Proof-tracking (Py, 1989).	Ability to actively assign both static and dynamic variable into the student model.
Reasoning with multiple diagnoses	Multiple expert metaphor, e.g. (Breuker, 1990; Self, 1988; Wenger, 1987).	Dealing with simultaneous presence of multiple events during instruction and diagnosis.

6.5 Summary

This chapter has discussed issues raised during the evaluation of GeNisa and the deficiencies inherent in ITS development methods. It has described a methodology and some of the design choices available to developers by postulating explicit theoretical and practical framework to address these problems and it has discussed the trade-offs involved. These trade-offs may be reconciled by the provision of improvement proposals, whose use may be governed by behaviour and contents. The methodology draws on experience gained during the design and implementation of GeNisa, and reviewed ITS and object oriented software engineering literature. Design requirements for developing ITS are also proposed. These design requirements are specified at different levels to corresponding to the standard ITS architectures.

However, the importance of this chapter goes beyond providing implementation approaches and guidelines for developing ITS. The main research point was to assert the approach for ITS development from reusable components models.

CHAPTER 7

SUMMARY AND CONCLUSIONS

7.1 Introduction

This chapter presents a summary of this research and provides a review and findings of this thesis. It draws out the main conclusions learnt during this study and approaches taken for the design and implementation of the generic architecture. Finally, directions for future research are proposed.

The next two sections provide a brief summary of each chapter, and present the major accomplishments of this research. This chapter concludes by discussing summary of contributions and the potential for future research arising from this study.

7.2 Summary of Chapter Contents

This thesis has investigated the design, implementation and evaluation of an intelligent architecture that provides a generic environment for ITS authoring and deployment. The aim was to investigate the feasibility of identifying and developing a set of components that can be reused in developing ITS, and to achieve components, modularity and portability. The research endeavour was investigated by combination of literature review, prototype development and empirical software evaluation. The focus of this research was the development of GeNisa, a generic environment for ITS development.

Chapter 1 introduces the motivation for this research and discusses standard ITS components followed by a discussion of the objectives for generic ITS. This chapter describes the ITS components and discusses them in the context of evolving application areas, and technology.

Chapter 2 presents reviewed research material and highlights particular limitations arising from the reviewed literature, with a particular focus on ITS component reusability and portability. The main ITS architectures are examined, and issues restricting reusability and portability are

identified for further investigation. This is followed by a critique of reviewed literature. Research material in related disciplines is presented to broaden the context of this research.

Chapter 3 introduces the research assumptions and discusses the theoretical aspects of generic architecture, and addresses limitations arising from the reviewed literature. Issues identified in Chapter 2 are examined and a perspective of the generic architecture is described. It is noted that generic ITS must provide for common authoring and interactive learning environment that are domain-independent and are reusable in different domains and across platforms. This chapter also discusses the requirements for GeNisa and the knowledge representation approach. It discusses the conceptualisation of components of the generic architecture including knowledge representation, interactive strategies and an automated knowledge acquisition. This chapter also discusses abstract implementation of the generic architecture, and issues such as portability and reusability are addressed.

Chapter 4 discusses the detailed architecture, design and implementation of GeNisa. It describes the composition of the system modules that this research has arrived at from the abstract design described in Chapter 3. An object-oriented class library has been developed in order to satisfy the requirements for the generic architecture discussed in Chapter 3. The class library also serves as a template to abstract commonality between similar components' classes, and implements the interactive features discussed in Chapter 2 and Chapter 3. This chapter also discusses contents and behaviour of the components of GeNisa application development and learning environments. This demonstrates the feasibility of a generic architecture that addresses the issues identified in Chapter 2 and the objectives of this research. A case-based tutorial for simulation modelling has been developed, to demonstrate how the theoretical descriptions discussed in Chapter 3 may be realised. The development and the learning environments have been implemented in Java.

Chapter 5 critically examines the design, implementation and evaluation of the GeNisa architecture, and further identifies strengths and weaknesses of this research. GeNisa has been evaluated on both Windows and Unix operating systems by means of formative and heuristic evaluation. The generic architecture has also been examined theoretically, and according to software engineering principles. The implementation (discussed in Chapter 4) has been appraised in order to examine if it poses any barrier to reusability and portability. Issues identified during evaluation were analysed to identify areas where specific functionality may be required.

Chapter 6 reflects on the journey through this study and discusses theoretical solutions to issues identified during evaluation (discussed in Chapter 5). On the basis of this research findings, this

chapter proposes a methodology for ITS development. The proposed methodology addresses the shortcomings identified from the evaluation of the generic architecture in Chapter 5. Furthermore, the original requirements for the generic architecture are reviewed and synthesised to identify areas for further improvement. This chapter concludes with improvement proposals for ITS. It is believed that these improvement proposals will bring about the much-needed development of shareable and reusable ITS components.

7.3 Conclusions

This research began by establishing a comprehensive background theory involving ITS (described in Chapter 2), which helped to identify problems and issues that are feasible for further investigation.

The reviewed literature has been critically examined in order to identify the problems addressed by published literature, design and implementation difficulties, platform limitations and components reuse capabilities. Particular focus has been placed on issues such as portability and reusability of ITS components. Based on a literature review, this study has first contrasted the different implementation methodology and application domains in order to identify the criteria that could be used to investigate the feasibility of ITS component reusability and portability.

The basic ITS architecture (Wenger, 1987) has been examined in some depth to elicit various strategies for improving modular construction and classifying interdependencies among components of the architecture. Each component of the architecture has been partitioned along the dimensions of context and behaviour in an attempt to make it more amenable to both portability and reusability. Therefore, it seems feasible to investigate the potential advantages of a generic architecture for ITS in order to reduce the time and the effort required for development.

The following sections summarise the major conclusions drawn from this study and examine the key issues investigated by this research. These conclusions relate to the research objectives described in Chapter 1.

7.3.1 Components Portability and Reusability

Based on a literature review (discussed in Chapter 2), a consolidated list of elements that are often integral features of ITS components were identified and analysed. Each ITS component was

taken in turn to identify reusable features. Each component feature has special attributes, which characterises its reusability and portability.

The literature review has also demonstrated that various ITS have employed techniques to increase reusability and portability of ITS components. These techniques have been helpful in the conceptualisation of the generic architecture (described in Chapter 3). The main purpose of generic architecture is to allow reusability and portability of ITS components by making explicit those properties and reasoning subsystems. The description of the GeNisa system in Chapter 4 illustrates the adaptive functionalities of the components of the development/learning environment. Furthermore, components class libraries have been used to address the issues of portability and reusability of ITS. This approach contrasts with much of the published literature in this area, where emphasis is often on instructional contents and pedagogical activities instead of reusability and portability (Murray, 1997). Although component portability and reusability has been the main focus of this research. There are two principal ways to address these issues: firstly, development of better class libraries that can reduce the need for developing new components/classes and requires less adaptation; secondly, the use of an architecture that is based on a formal, abstract model of system behaviour can provide a practical means of describing and analysing components, and interaction within the system. Both frameworks have been pursued in this research.

GeNisa architecture has been implemented in Java (Arnold et al., 2000). Some procedures were implemented (discussed in Chapter 4) to isolate platform dependence; for example all system-dependent calls have been abstracted to platform-independent procedures, and are contained in a class library, which can be accessed indirectly by different range of the programs. This framework allows this study to accomplish most of the work on portability of the framework to another platform by simply replacing the platform-dependent library. Furthermore, GeNisa components have a high level of abstraction, which allows each component to be considered as a self-contained, modular and shareable object, thereby allowing GeNisa class libraries to be utilised as COM compliance. COM provides enormous flexibility for cross-platform utilisation of different range of components, and allows other tools to be integrated irrespective of the domain and knowledge representation formalism. Templates (discussed in Chapter 4) have been used to abstract commonality between similar classes, content and interactions. Furthermore, developing components class libraries is a process of making increasingly detailed components behaviour and interactions. To facilitate this process, an ontological representation of the component model should be formulated explicitly. This approach may promote reuse because existing components

(contents and behaviour) can be used as components of larger application. This could improve scalability because it may require less effort by a developer to modify and maintain.

7.3.2 Knowledge Representation

The generic architecture has a particular underlying representational framework, and therefore includes its own assumptions about ITS components, and it embodies constraints on the types of components that it can build.

An abstract design of the knowledge base is discussed in Chapter 3. This knowledge representation formalism uses ontology for conceptualising the objects and relationships in a domain (Chandrasekaran et al., 1999), (Gruber, 1993), (Gruber, 1995) and to represent its problem-solving methods (Musen et al., 1995). As currently implemented, ontology has been used to specify component types, link types, properties, and behaviour. This framework may enable domain experts to construct, test, and modify application-specific knowledge rapidly. The key to this approach is that it permits the developer to utilise knowledge in a variety of formats, and allows them to transform the knowledge expressed in these different formats into constraints to be utilised during instruction.

On the basis of the discussion in Chapter 3 and 4, it seems feasible to infer that using a flexible knowledge representation formalism developers can create a range of ITS with minimal cost (Murray, 1999). The GeNisa architecture has been designed and implemented using appropriate knowledge representation formalism in order to achieve both scope and usability of ITS (Murray, 1998).

Furthermore, in order to maximise the leverage of the knowledge representation formalisms, it is important that the knowledge representation method help the student integrate his/her knowledge into a coherent model of the domain and solve problems in the domain. The key feature of this approach is that the developer could use multiple models of the domain in order to facilitate knowledge acquisition. Also, the use of predefined model of knowledge for prototypical tasks, and the dynamic invocation of these tasks can guide ITS development of process. This may lead to a reduction in the number of parameters that has to be instantiated during implementation.

This formalism of knowledge abstraction may promote a degree of independence among a range of components, and therefore offers the possibility of being able to share and reuse these components.

7.3.3 The Generic Architecture

In general, the purpose of developing generic system architectures is to discover high-level frameworks for reusing ITS components, their subsystems, and their interactions with related systems. The generic architecture is not a blue print for designing a single system, but a framework for designing a range of systems over time, and for the analysis and comparison of these systems. By revealing the shared components of different systems at the right level of generality, the generic architecture may promote the design and implementation of components and subsystems that are reusable, cost-effective and adaptable. Furthermore, the main goal of an ITS authoring system is to make the process of building an ITS easier (Murray, 1998). This ease should translate into reduced cost, development life cycle and decrease in the skill required for potential developers.

This research has focused on synthesising component-based approach, object oriented method and AI by combining independent components to support modularity, portability, and extendibility. The outcome of the abstract design (discussed in Chapter 3) is a GeNisa architecture for ITS, consisting of design and learning environments (discussed in Chapters 4). Each environment consists of modular components. Each component is divided into two major subsystems i.e. contents and behaviour. Component contents are active processes that communicate and co-ordinate users' activities in order to create the required intelligent behaviour. The behaviour subsystem co-ordinates all components' events processes.

The generic architecture has demonstrated, in Chapter 4, a well-structured and transparent implementation of conceptually complex AI components and reasoning processes. Courseware developed with this framework can be adapted by modifying their component library and elaborating functionality of each of the component subsystem. Also, the choice of development language, operating system, and knowledge representation formalism may affect reusability and portability. This should translate into development life cycle and decrease in the skills required for developers.

To satisfy requirements for heterogeneous platforms (discussed in Chapter 3), the generic architecture has been implemented to enable usage over the Internet and Intranets. This was exemplified in Chapter 4. The generic architecture components may be utilised across a variety of platforms, with a portable source code. This approach precludes the use of platform-specific methodologies and provides a more flexible, cost-effective and scalable solution to ITS development. Besides the expected extensibility that comes with the object oriented programming

paradigm, the generic architecture is easily extensible by using component constraints and library pattern.

Component constraints have been implemented as instances of an abstract class. Arbitrary constraints may be easily added to the system by sub-classing and adding an instance of the new component class to the global library. Thus, it is relatively easy to extend the system using the abstract class with very minimal disruption to the program code.

Based on the literature review (Chapter 2), design and implementation of GeNisa (Chapter 3 and 4) and evaluation in Chapter 5, if could be inferred that the GeNisa architecture design is flexible in its implementation of its intrinsic model. The model is very easy to modify because most of the components of the architecture, such as user interface, inference engine, design assistant, wizards, etc, have component consistency constraints, which can easily be modified as required. These constraints exist as part of the component library and are added to the base type objects (menus, dialogue and toolbars, scrollbar) during program initialisation. This framework makes it easy to remove or replace these constraints, thereby promoting reusability.

The generic architecture design and implementation framework used for this research, has demonstrated the feasibility for a generic architecture for ITS. There are two major advantages of this framework. First, since it is based on a component-based approach and object oriented software-engineering methodologies, the framework provides a systematic approach that can be used to develop a range of ITS components for real world utilisation. Second, contents of the learning environment can be used across domain and across platforms.

Integrating the components of the generic architecture and the knowledge bases requires consideration of the behaviour and functionalities of each component. Sharing knowledge structures, interaction mechanism and using common interfaces among the different components may help to maintain modularity and flexibility, thereby, simplifying system maintenance and development.

7.3.4 Facilitating Design Processes

One of the tenets of this research is that all conceptual and structural elements of a representational formalism must be represented graphically using appropriate visual components. This approach is essential if an ITS development environment is to be used by non-programmers without the need for a knowledge engineering expertise (Murray, 1998). This approach is

different from approaches discussed in Chapter 2 because reuse of existing components, even if they require adaptation, can reduce the overall costs of ITS development. Furthermore, the components representation formalism outlined in this study has a deeper and more general representation of the task, which are reusable and portable across platforms.

This research has demonstrated (Chapter 3 and 4) that ITS are complex systems containing embedded models of several components such as domain models, inference engine, pedagogical and student models. The authoring processes investigated in this research may reduce the design process to a sequence of tasks. This approach is feasible by separating component contents and their behaviour, plus decomposing content in a way that maintains coherence and consistence when it is reconstructed and use during instruction. This approach enables the architecture to be flexible and modular. The generic architecture framework has the potential to increase the efficiency of building ITS by reusing common components elements (Murray, 1999). The implementation methods (discussed in Chapter 4) and subsequent evaluation in Chapter 5, illustrated the object-oriented and component-based approach used for implementation.

The class libraries and separation of components contents and their underlying behaviour may allow components to be organised into different levels of abstraction. This framework may help identify several "refinement" layers between component behaviour and their events. Furthermore, it also allows the generic architecture to be represented as a heterogeneous integration of subsystems providing a wide range of components functionalities. Although the GeNisa development environment can scaffold both the underlying representational structure and the design process itself, the developer will need to have some understanding of both the representational structure and the design process. Some degree of object-oriented modelling skills will be helpful.

Chapter 6 highlights some practical issues associated with the design and implementation of the GeNisa prototype, and proposes a methodology to address these difficulties. The proposed methodology defines approaches and protocols for design and implementation of ITS. However, the framework does not prescribe how developers must implement the base component functionality that is suggested. This proposal may serve as a starting point for ITS development and researchers can consult the proposal in order to derive specific design rules appropriate for a particular system, irrespective of their implementation platforms. This approach may make ITS design processes more efficient. Furthermore, the proposed framework may be used for designing a range of ITS components and reusing existing ones.

The key issues highlighted during evaluation of GeNisa are component reusability and cross-platform support. On the basis of the experience gained in this research, and on current ITS literature, this research has delineated improvement proposals and requirements for enhancing ITS development. The requirement proposals can help establish firmer rules for co-ordinating individual ITS design activities and implementation. They can also help to make design decisions quicker and they facilitate defining detailed design requirements. The improvement proposal allow for: (i) a more efficient implementation of the ITS because the common components and interfaces are only implemented once and have been abstracted into a class library, (ii) the system to adapt to new hardware and software changes because the adaptation is only incremental when viewed at the right level of abstraction. Theoretically, each of these elements can exist as a portable autonomous component, and can be reused across platforms. Furthermore, the major advantage of the improvement proposal is that it allows the developer to integrate new knowledge into a coherent object-oriented model of the domain, and thereby facilitate knowledge integration and reuse.

There is an efficiency cost associated with maintaining the implementation of the methods outlined in this research. Reducing computational complexity could reduce the overall development lifecycle significantly, and this may be necessary in the future if the framework is applied to a larger domain or is extended to different kinds of platforms. Furthermore, the framework could accelerate the development of ITS by utilising an existing reusable library and support the delivery of a tutorial on different platforms.

Based on the above discussion and on issues articulated in previous Chapters, the main conclusions of this research are synthesised and are summaries in the Table 7.1. The table also relates aspects of the research objectives to the conclusions drawn.

7.3.5 Summary of Conclusions/Contributions

The major contributions of this research are shown in Table 7.1. The objectives (i) to (xi) are those given in Chapter 1 on page 15.

Table 7.1 Research Objectives and Outcomes/Contributions

RESEARCH OBJECTIVES	CONCLUSIONS
i. To review critically ITS published literature, with a particular focus on ITS components, reusability issues and interactivities.	Continuing research in the design and implementation of ITS has produced a large collection of ITS, which may improve instruction. However, some of these systems exist as domain specific. This research shows that ITS has not yet achieved widespread availability and use, nor impact on mainstream teaching and learning. This may be attributed to the lack of a skilled intelligent component developer; and ITS is closed and monolithic, which may prevent reuse, extension, or modification of the system except by its original developers. ITS development requires expensive programming skills. To address these problems, the research has investigated and demonstrated the feasibility of developing modular and reusable techniques for developing ITS. Using such techniques, a unit of ITS component functionality can be expressed independently of the application in which it is used. In this approach, component entities become reusable in multiple environments without having to be re-implemented. The GeNisa framework demonstrates an attempt towards a more unified view of developing ITS. This research shows that literature on the ITS components portability and reusability is limited and sometimes characterised by anecdotal evidence from a variety of less formal development methodology/methodologies.
ii. To propose a methodology for Intelligent Tutoring Systems Development.	This research shows that there are developmental advantages to be realised by using shared content, including media, pedagogical strategies, and intelligent behaviour. In particular, components should be defined in such a way that they check inter and intra type inter-component consistency constraints across application. Components may embed automatic constraint preservation in terms of change propagation, if appropriate. The proposed methodology should ensure persistence of changes made during execution of the system. This approach could preserve user effort against hardware and software failures. Also, it makes changes visible to concurrent users as soon as tool commands have been completed. Each component of the proposed methodology, therefore, offers a set of well-defined generic behaviour that the event process manager can use to perform certain interactive activities across platforms. Incorporating component reuse into ITS development may provide programming leverage, because such components source code can easily be scaled beyond the component threshold. Conventional technology such as ActiveX, DCOM, CORBA and JavaBeans offers a platform that contains the needed mechanism in the form of component-based architecture, which has the potential to provide reusability, and portability of ITS. GeNisa facilitates design of ITS by using appropriate interactive features and careful utilisation of graphical objects. Furthermore, it will be useful to have some ITS authoring tools providing features, which allow a smooth transition from traditional authoring paradigms to authoring more powerful intelligent components. This research show that, as demonstrated in Chapter 4, separation of authoring and development environment can improve compatibility between components because of interface conventions, which can assist in the integration process. This may also result in programs which have improved designs, are better understood, have better interactivities and are more robust.
iii. To develop a prototype of a	On the basis of the experience gained during this research, it seems

multi-platform learning and authoring environment, reusable for different scenarios and applicable to other domains.	feasible to infer that the advantages of a generic architecture will become increasingly attractive to ITS developers and researchers. This is because ITS needs to be able to reuse its components in order to address the issues discussed in Chapter 1. ITS component reuse across domains and platforms may result in considerable efficiencies and a more useful components development. This research has demonstrated that the principles of software engineering, object oriented methods may be used to address the issues of knowledge acquisition bottleneck in ITS. However, the principles of software engineering and object-oriented methods alone do not cover the whole of ITS development; principles from cognitive psychology, expert systems, instructional system development theories, case-based reasoning must be included. Object-oriented class libraries offer the potential developers an approach to manage range of components and to be used for a range of purposes. This approach allows the system's components to be represented in terms of a heterogeneous collection of components providing a wide range of application functionalities.
iv. To investigate how ITS can use case-based reasoning to represent knowledge and use those representations to monitor and reason about user learning processing and to guide remedial learning in order to improve the instructional processes.	This study showed that the integration between case-based approach and ITS can be explained in terms of knowledge structures. Case-based approach is to help the student integrate his/her knowledge into a coherent qualitative causal model of the domain and solve problems in the domain. The key feature of this model is that the tutor may use multiple models of the domain in order to facilitate knowledge integration. Furthermore, using both rules for retrieving cases during instruction might help achieve a higher accuracy level than using cases alone. This shows that the combination of rule-based and case-based reasoning can improve the learning with a case-based reasoning. This research has demonstrated that component-based approach and AI techniques can be applied to the design and implementation of simulation modelling courseware (discussed in Chapter 4). The instructional components of the courseware are modular and are reusable across domains. Furthermore, the development of adaptive courseware should accommodate users with very different backgrounds, prior knowledge of the subject and learning objectives and should guide the user adaptively through the course; and must include the necessary instructional factors. This research has demonstrated that using case scenarios may enhance the acquisition of simulation knowledge and the development for different problem-solving skill. Specifically, using self-explanation strategies during instruction could stimulate and scaffold the acquisition of domain knowledge. As demonstrated in previous chapters, components-based approach offers several advantages compared to conventional ITS development. The design and implementation of the GeNisa architecture demonstrates that this framework may result in much simpler implementations than other existing techniques and it can offer greater degree of reusability. This research shows that component-based approach is a useful software engineering technique, and has the potential to greatly facilitate rapid application development and code reusability. It should promote components modularity by using an object-oriented approach for developing ITS and standardisation of components development processes.
v. To evaluate critically, and empirically, the effectiveness of the authoring and learning environments implemented.	The conceptual and structural elements of a representational formalism should be depicted graphically with high visual objects if ITS authoring systems are to be used by non-programmers. This approach may relieve working memory load by providing explicit

	sub-goals, plans, rules and to understand about the underlying structure of the domain problems. To achieve greater flexibility, and reusability, ITS should include the ability to customise the representational formalism as required. Furthermore, the empirical study revealed that courseware should support learning at different stages of instruction and must include the necessary instructional factors. The results of this evaluation indicated that ITS should foster students engagement and learning and allow students to develop a constructive approach to their learning. GeNisa may facilitate the design of ITS by using appropriate interactive features and careful utilisation of graphical objects. As demonstrated in Chapter 4, separation of authoring and development environments can improve compatibility between components because of interface conventions, which can assist in the integration process. This may also result in program which have improved designs, are better understood, better interactivities and are more robust. This research has demonstrated that interactivity factors should discriminate between good and bad quality component contents and behaviour. Essentially, a balance of emphasis for both components and instruction should be emphasised at various stages of ITS development. This balance may be difficult to realise because of limited expertise for instructional/component development. Instructional content should be represented and authored by using knowledge representation formalisms so that it can be reused across domains. This research has demonstrated that formal specification and requirements of ITS may provide an approach for establishing ontological formalisms for formalising ITS components development processes. On the basis of discussion in Chapter 3 and 4, it can be inferred that incorporating intelligent components into instructional environment should enable a greatly improved student model and may provide a better pedagogical expert, plus a more effective remedial advice. A variety of component functionalities and methods should be employed by developers in order to support the development of interactive instruction and allow them to be more widely available.
vi. To identify required future developments in ITS.	The further developmental requirements identified in Chapter 6 may leverage ITS development by defining components in smaller contexts, which may maximise abstraction, reusability and semantic coherence. This proposals as shown that the elicitation of the intrinsic components functionalities, the use of OO and AI methodologies may help to define common mechanisms, which could be used to elicit and organise knowledge in a reusable and "transparent" way. Furthermore, the proposal places particular emphasis on modularity, and flexibility, which may facilitate code reuse. While these attributes may be required in any application, it could be used as a model for creating reusable authoring tools for ITS.

7.4 Further Research

The research reported in this thesis can be continued in several directions. The following areas for further research have been identified:

7.4.1 Instructional Factors

Further investigations can be carried out to test whether instructional factors used in GeNisa should be applied in empirical research to enable the comparison of findings and results between related studies.

Further research should also attempt to identify any other basic instructional factors that could enhance the approach developed in this thesis. Only subsets of behaviours of the learner during instruction have been considered for the expert model. It would be extremely advantageous to model other complex behaviours for GeNisa. For example, a study to investigate the nature of the integration between the domain and the pedagogy expert could further enhance the instructional environment.

7.4.2 Development Environment

This research has proposed a methodology for ITS development. Further research could be directed to investigate the generality of this approach to develop Bayesian models (developers problem solving methods) in other domains (such as electronic circuit design). The domain expert uses multiple models of the domain. It is believed that the models developed during this research are only a small set of the possible ones that the instructional module uses while performing reasoning in the domain. A definite research direction is to investigate other possible domain models and their usage during instruction.

One advantage of the generic architecture paradigm is that it has the potential for reuse and allows changes to be made easily. Future work in this area should include a reassessment of Common Object Request Broker Architecture (CORBA) to determine the extent to which this de facto framework for distributed object technology can benefit the field of ITS.

Object-oriented approaches to ITS development must be investigated further. Many features from the areas of object-oriented analysis, design and implementation can be extended to the ITS model. Integrating an object-oriented data model with ITS can greatly improve the information retrieval process. This can be accomplished by redirecting ITS functionality from the application level to the database level using object-oriented database systems.

The design of student modelling for GeNisa is based on qualitative reasoning. The student model assesses the student globally. This opinion may increase or decrease during tutoring. What

aspects of their behaviour are responsible for this? What aspects of their behaviour limit the effectiveness of tutoring? A fully implemented ITS architecture may help provide some answers to these questions.

The prototype implements the knowledge acquisition components of GeNisa, and develops the knowledge required to use a standard training model. An implementation of a complete instructional environment, covering the span from knowledge acquisition to final training, could provide a valuable insight as to the methodology used in GeNisa. Because of the magnitude of this task, a smaller domain could be used, or pre-existing knowledge components could be used if they were available. Additionally, the complete prototype could be implemented in other languages, allowing efficiency issues to be more fully explored. Authoring tools to support this type of knowledge sharing must be useable in heterogeneous platforms.

The study presented in this thesis, the conclusions drawn, and the recommendations stated, focused on the development and evaluation of GeNisa. The research assumptions and the objectives as outlined in the introductory chapters provided the starting-point for this investigation.

With further development, the design and implementation methodology postulated in this research could provide a promising method for augmenting a software engineering toolkit with a new technique for application in instructional development.

REFERENCES

Aamodt, A. and Plaza, E. (1994). Case-Based Reasoning: Foundational issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications*, 7 (1), pp.39-59.

Aben, M. (1993). Formally Specifying Reusable Knowledge Model Components. *Knowledge Acquisition*, 5, pp. 119 - 141.

Adiga, S. and Glassey, C. R. (1991). Object-oriented Simulation to Support Research in Manufacturing Systems. *International Journal of Production Research*, **29** (12), pp. 2529-2542.

Agius, H.W. and Angelides, M.C. (1997). Integrating Logical Video and Audio Segments with Content-related Information in Instructional Multimedia Systems. *Information and Software Technology*. **39** (10), pp. 679-694.

Akpinar, Y. and Hartley, J.R. (1996). Designing Interactive Learning Environments. *Journal of Computer Assisted Learning*, 12, pp. 33-46.

Allen, R. B. (1990). User Models: Theory, Method, and Practice. *International Journal of Man Machine Studies*, 32, pp. 511-543.

Ambler, S. W. (1998). *Building Object Applications That Work - Your Step-by-Step Handbook for Developing Robust Systems With Object Technology*. New York: Cambridge University Press.

Ambron, S. and Hooper, K. (1988). *Interactive Multimedia*. Microsoft Press, Redmond, WA.

Anania, J. (1983). The Influence of Instructional Conditions on Student Learning and Achievement. *Evaluation in Education*, 7, pp. 1-92.

Anderson, J. R. (1993). Problem Solving and Learning. *American Psychologist, 48,* pp. 35-44.

Anderson, J. R. (1992). Intelligent Tutoring and High School Mathematics. Frasson, C., Gauthier, G. and McCalla, G. I. (Eds.), *Intelligent Tutoring Systems*: Second *International Conference*, Springer-Verlag, Berlin, pp. 1-10.

Anderson, J. R. (1991). The Place of Cognitive Architectures in a Rational Analysis VanLehn, K., (Ed), *Architectures for Intelligence*, Lawrence Erlbaum, Hillsdale, NJ, pp. 1-24.

Anderson, J. R. (1990). Analysis of Student Performance with the LISP Tutor. Fredericksen, N. Glaser, J. Lesgold, A. and Shafto, M. (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*. Lawrence Erlbaum, Hillsdale, NJ, pp. 27-50.

Anderson, J. R. (1988). The Expert Module. Polson, M. C. and Richardson, J. J. (Eds.), *Foundations of Intelligent Tutoring Systems*. Lawrence Erlbaum, Hillsdale, NJ, pp. 21-53.

Anderson, J. R. (1987). Skill Acquisition: Compilation of Weak-Method Problem Solutions. *Psychological Review,* 94, pp. 192-210.

Anderson, J. R. (1983). *The Architecture of Cognition.* Harvard University Press. Cambridge, MA.

Anderson, J. R., Boyle, C., and Reiser, B. (1985). Intelligent Tutoring Systems. *Science*, 228, pp. 456-462.

Anderson, J. R., Conrad, F. G., and Corbett, A. T. (1989). Skill Acquisition and the LISP Tutor. *Cognitive Science*, **13** (4), pp. 467-505.

Anderson, J. R., Corbett, A., Fincham, J., Hoffman, D., and Pelletier, R. (1992). General Principles for an Intelligent Tutoring Architecture. In Shute, V. and Regian, W. (Eds.), *Cognitive Approaches to Automated Instruction*, Lawrence Erlbaum, Hillsdale, NJ, pp. 81-106.

Anderson, J. R., Corbett, A.T., Koedinger, K. R. and Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences*, *4*, pp.167-207.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences*, **4** (2), pp. 167-207.

Anderson, J. R. and Pelletier, R. (1991). A Development System for Model-Tracing Tutors. *In Proceedings of the International Conference of the Learning Sciences*, Evanston, IL, pp. 1-8.

Anderson, J. R., and Reiser, B.J. (1985). The LISP tutor. *Byte, 10*, pp.159-175.

André, E., Rist, T., and Müller, J. (1999). Employing AI Methods to Control the Behaviour of Animated Interface Agents. *Applied Artificial Intelligence*, 13, pp. 415-448.

Anglin, G. (Ed.) (1995). *Instructional Technology, Past, Present, Future.* Co., Libraries Unlimited, Englewood.

Angelides M. C. and Gibson G. (1993). Pedro - The Spanish Tutor: A Hypertext-based Intelligent Tutoring System for Foreign Language Learning. *Hypermedia*, **5** (3), pp. 205-230.

Angelides, M.C. and Paul, R.J. (1993). Developing an Intelligent Tutoring System for a Business Simulation Game. *Simulation Practice and Theory*, **1** (3), pp.109-135.

Angelides, M.C and Paul, R.J. (1995). Providing Intelligent Tutoring Systems within a Gaming-Simulation Environment for Learning. *Journal of Intelligent Systems* **5** (2-4), pp. 319-350.

Angelides, M.C. and Stanley, A. (1994). Towards a New Dimensions in CBT. Thoughts in the Retail Industry. Brunnsteinand, K. and Raubold, E., (Eds.), *Applications and Impacts, Information Processing*, Elsevier Science, North Holland, pp. 639-644.

Angelides, M.C. and Tong, A.K.Y. (1995). Implementing Multiple Tutoring Strategies in an Intelligent Tutoring System for Music Learning. *Journal of Information Technology,* **10** (1), pp. 52-62.

Apple Computer Inc. (1997). *Macintosh HyperCard User's Guide*. Apple Corporation.

Apple Computer Inc. (1993). *Macintosh Human Interface Guidelines*. Addison-Wesley, Reading, MA.

Arnold, K., Gosling, J., Holmes, D. (2000). *The Java Programming Language*. Third Edition. Addison-Wesley, Reading, MA.

Arruarte, A., Fernandez-Castro, I., Ferrero, B. and Greer, J. (1997). The IRIS shell: How to Build ITS from Pedagogical and Design requisites. *International Journal of Artificial Intelligence in Education*, **8** (3-4), pp. 341-381.

Arshad, F.N and Kelleher, G. (1990). Learning with Computer-Based Advisor. *Psychological Research Review*, 9, pp. 130-139.

Ashley, K. (1991). Reasoning with Cases and Hypothetical in HYPO. *International Journal of Man-Machine Studies*, 34, pp. 753-796.

Atolagbe, T. and Hlupic, V. (1998). A Conceptual Model for An Internet Based Intelligent Tutoring System for Simulation Modelling. *In Proceedings of 10th European Simulation Symposium and Exhibition*. Bargiela, A. and Kerckhoffs, E. (Eds.), Nottingham, pp. 692–694.

Atolagbe, T. (1997). A Generic Architecture for Intelligent Instruction for Simulation Modelling Software Packages, *Informatica,* 21, pp. 643–650.

Atolagbe. A. T. and Hlupic, V. (1997). A Reusable Architecture for Intelligent Tutoring Systems for Teaching Simulation Modelling. *In the Proceedings of the 9th European Symposium on Simulation in Industry Conference*. Kaylan, A.R. and Lehmann, A. (Eds.) Passau, pp. 187–192.

Atolagbe, T. and Hlupic, V. (1997). A Generic Architecture for Intelligent Instruction for Simulation Modelling Software Packages, *In the Proceedings of Winter Simulation Conference*, Charnes, J.M, Morrice, D.J., Brunner, D.T., and Swain, J.J., (Eds.), pp. 856–863.

Atolagbe, T. and Hlupic, V. (1997). SimTutor: A Multimedia Intelligent Tutoring Systems for Simulation Modelling. *In the Proceeding of the Winter Simulation Conference*. Andradottir, S., Healy, K. J.. Withers, D. H and Nelson, B. L. (Eds.), Georgia, pp. 504–509.

Atolagbe, T. and Hlupic, V. (1997). Interactive Strategies for Developing Intuitive knowledge as Basis for Simulation Modelling Education. *In the Proceeding of the Winter Simulation Conference*. Andradottir, S., Healy, K. J., Withers, D. H., and Nelson, B. L. (Eds.), Georgia, pp. 1394-1402.

Atolagbe, T. and Hlupic, V. (1997). Intelligent Multimedia Tutoring for Simulation Modelling Education. *Proceeding of the European Simulation Symposium*, Istanbul, Turkey, Kaylan, A.R. and Lehmann, A. (Eds.), pp. 280-284.

Atolagbe, T. and Hlupic, V. (1996). A Generic Architecture for Intelligent Instruction for Simulation Modelling Software Packages, *Proceedings of Winter Simulation Conference*, Charnes, J. M, Morrice, D. J., Brunner, D. T., and Swain, J. J., (Eds.), pp. 856-863.

Atolagbe, T. and Hlupic, V. A Generic Architecture for Instructional Systems for Simulation Modelling. (1996). *Proceedings of European International Conference on Simulation Modelling,* Javor, A., Lehmann, A., and Molnar, I. (Eds.). Budapest, pp. 389-393.

Ausubel, D. (1978). In Defense of Advance Organisers: A Reply to the Critics. *Review of Educational Research*, 48, pp. 251-257.

Ausubel, D., Novak, J., and Hanesian, H. (1978). *Educational Psychology: A Cognitive* View (Second Edition). Rinehart and Winston, New York.

Avgerou, C. and Cornford, T. (1998). *Developing Information Systems - Concepts, Issues and Practice*, Second edition, Macmillan Press, Basingstoke, UK.

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., van Dantzich, M., and Wax, T. (1997). Lifelike Computer Characters: The Persona Project at Microsoft. In Bradshaw, J., (Ed.), *Software Agents*. AAAI/MIT Press, Menlo Park, CA.

Baddeley, A. D. (1994). Working Memory. In M. S. Gazzaniga (Ed), *The Cognitive Neurosciences*, Cambridge, Mass.: MIT Press. pp. 755-764.

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning*. Academic Press, Boston.

Barker, P.G., van Schaik, P. and Hudson, S.R.G., (1998). Mental Models and Lifelong Learning, *Innovations in Education and Training International*, **35** (4), pp. 310-318.

Barr, A. and Feigenbaum, E. A. (1982). Applications-Oriented AI Research: Education. *Handbook of Artificial Intelligence*, Vol. II. Addison-Wesley, Reading, MA.

Barrett, S. E., and Shepp, B. (1988). Developmental Changes in Attentional Skills: The Effect of Irrelevant Variations on Encoding and Response Selection. *Journal of Experimental Child Psychology*, 45, pp. 382-399.

Batini, C., Lenzerini, M. and Navathe, S. B.(1986). A Comparative Analysis of Methodologies for Database Schema Integration, *ACM Computing Surveys*, **18** (4), pp.323 - 364.

Beaumont I (1994) User modelling in the Interactive Anatomy Tutoring System: ANATOM-TUTOR. *User Models and User Adapted Interaction*, **4** (1), pp. 21-45.

Beeson, M.J. (1990). Mathpert: A Computerised Learning Environment for Algebra, Trigonometry, and Calculus, *International Journal of Artificial Intelligence in Education*, **1** (4), pp.65-76.

Bell, B. (1998). Investigate and Decide Learning Environments: Specialising Task Models for Authoring Tools Design. *Journal of the Learning Sciences*, Lawrence Erlbaum, Hillsdale, NJ, **7** (1), pp. 65-106.

Benedikt, M. (1991). *Cyberspace*: *First Steps*. The MIT Press, Cambridge, MA.

Benjamins, V. R. (1995). Problem Solving Methods for Diagnosis And Their Role in Knowledge Acquisition. In *International Journal of Expert Systems: Research and Application*, 8 (2), pp. 93-120.

Benjamins, V. R. and Fensel, D. (1998). The Ontological Engineering Initiative $(KA)^2$. In Guarino, N. (Ed.), Formal Ontology, *Information Systems*, IOS Press, pp. 287-301.

Benjamins, V. R., Fensel, D., Decker, S., and Perez, A. (1999). $(KA)^2$: Building Ontologies for the Internet: A Mid Term Report. In the *International Journal of Human-Computer Studies*, 51, pp. 687-712.

Bennett, J.S. (1989). ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert system. *Automated Reasoning*, 1, pp. 49-74.

Benysh, D. V., Koubek, R. J., and Calvez, V. (1993). A Comparative Review of Knowledge Structure Measurement Techniques for Interface Design. *International Journal of Human-Computer Interaction*, **5** (3), pp. 211-237.

Berners-Lee, T. (1996). HTTP: A Protocol for Networked Information. http://sunsite.doc.ic.ac.uk/rfc/rfc1945.txt

Berners-Lee, T. and Conolly, D. (1995). *Hypertext Mark-up Language Specification 2.0*. http://sunsite.doc.ic.ac.uk/rfc/rfc1866.txt.

Bielaczyc, K., Pirolli, P., and Brown, A. L. (1995). Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem-Solving. *Cognition and Instruction*, **13**(2), pp.221-252.

Biggerstaff, T. J. and Perlis, A. J. (1989), *Software Reusability*, Volumes 1 and 2, Addision Wesley/ACM Frontier Series, Reading, MA.

Biggerstaff, T. J. and Richter, C. (1987). Reusability Framework, Assessment, and Directions, *IEEE Software*, **4** (2), pp. 41-49.

Bloom, B. (1956). *Taxonomy of Educational Objectives*. Mackay Publishing, New York.

Bloom, B. S. (1984). The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, **13** (6), pp. 4-16.

Boehm, B.W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*, pp. 61-72.

Bonnardel, N. and Sumner, T. (1996). Supporting Evaluation in Design: The Impact of Critiquing Systems on Designers of Different Skill Levels. *Acta Phychologica*, 91, pp. 221-244.

Booth, P. (1989). *An Introduction to Human-Computer Interaction*. Lawrence Erlbaum, London.

Booch, G., Rumbaugh, J. and Jacobson, I., (1999a). *The Unified Modelling Language User Guide*. Addison Wesley, Reading, Mass.

Booch, G., Rumbaugh, J. and Jacobson, I., (1999b). *The Unified Modelling Language Reference Manual*. Addison Wesley, Reading, Mass.

Booch, G., (1998). *Architecture Patterns*. Rational Software User's Group, Orlando, FL. 1998.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Benjamin-Cummings, Redwood City, CA.

Borko, H, Livingstone, C., MacCaleb, J. and Mauro, L. (1988). *Calderhead Teachers' Professional Knowledge*. Farmer Press.

Box, D. (1998). *Essential COM*. Addison Wesley.

Brachman, R. J. and Levesque, H. J. (1985). Readings in Knowledge Representation. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Breuker, J.A. (1994). A Suite of Problem Types. In Breuker, J. and Van de Velde, W. (Eds.), *CommonKADS Library for Expertise Modeling: Reusable Problem Solving Components*, IOS Press, Amsterdam, pp. 57-87.

Breuker, J. (1990). *Conceptual Model of Intelligent Help System*. Breuker, J. (Ed.), *EUROHELP: Developing Intelligent Help Systems*. Copenhagen, pp. 41-67.

Breuker, J. (1988). Coaching in Help Systems. In Self, J. (Ed.), *Artificial Intelligence and Human Learning*. Chapman and Hall Computing, London, pp. 310-337.

Breuker, J.A., and van de Velde, W. (1994). *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components*, IOS Press, Amsterdam.

Brooks, J. G. (1993). Search of Understanding: *The Case for Constructivist Classrooms*. Association for Supervision and Curriculum Development, Alexandria, VA.

Brown, J. S. (1990). Towards a New Epistemology of Learning. Frasson, C. and Gauthier, G. (Eds.), *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Ablex Publishing, Norwood, NJ, pp. 266-282.

Brown, J. S. and Burton, R.R. (1978). Diagnostic Models For Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 2, pp.155-191.

Brown, J. S., and Burton, R. R. (1975). Multiple Representations of Knowledge for Tutorial Reasoning. Bobrow D. G. and Collins, A. (Editors.), *Representation and Understanding*. Academic Press, New York, pp. 311-349.

Brown, J. S., Burton, R. R. and deKleer, J. (1982). Pedagogical Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III. Sleeman, D. and Brown, J.S. (Eds.), *Intelligent Tutoring Systems*. Academic Press, New York, pp. 227-282.

Brown, J. S., Collins, A., and Duguid, P. (1989). Situated Cognition and the Culture of Learning. *Educational Researcher,* **18** (1), pp. 32-42.

Brown, S. I. and Walter, M. I. (1990). *Problem Posing*. Lawrence Erlbaum, Hillsdale, NJ.

Bruffee, K. (1993). *Collaborative Learning. Higher Education, Interdependence, and the Authority of Knowledge*. The Johns Hopkins University Press, Baltimore.

Bruner, J. (1990). *Acts of Meaning*. Harvard UniversityPress, Cambridge, MA.

Bruner, J. S. (1966). *Towards a Theory of Instruction*. Norton and Co., New York.

Bruner, J. S. (1961). The Act of Discovery. *Harvard Educational Review*, **31** (1), pp.21-32.

Brusilovsky, P. (1995a). Intelligent Learning Environments for Programming: The Case for Integration and Adaptation. Greer, J. (Eds.), *Proceedings of Seventh World Conference on Artificial Intelligence in Education*, Washington, DC. pp. 1-8.

Brusilovsky, P. (1995b) Intelligent tutoring systems for World-Wide Web. In. Holzapfel, R (Ed.) *Proceedings of Third International WWW Conference*, Darmstadt, Darmstadt, Fraunhofer Institute for Computer Graphics, pp. 42-45.

Brusilovsky, P. (1998). Methods and Techniques of Adaptive Hypermedia. Brusilovsky, P. Kobsa, A. and Vassileva, J. (Eds.). *Adaptive Hypertext and Hypermedia* Kluwer Academic Publishers, Dordrecht, pp. 1-43.

Brusilovsky, P. (1997a) Efficient Techniques for Adaptive Hypermedia. Nicholas, C. and Mayfield, J. (Eds.), *Intelligent Hypertext: Advanced Techniques for the World Wide Web. Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1326, pp. 12-30.

Brusilovsky, P. (1997b). Integrating Hypermedia and Intelligent Tutoring Technologies: From Systems to Authoring Tools. Kommers, P., Dovgiallo, A., Petrushin, V. and Brusilovsky, P. (Eds.) *New Media and Telematic Technologies for Education in Eastern European Countries*. University Press, Twente Enschede, pp. 129-140.

Brusilovsky, P. (1996b). Methods and Techniques of Adaptive Hypermedia. *User Modelling and User-Adapted Interaction*. **6** (2-3), pp.87-129.

Brusilovsky, P. and Cooper, D. W. (1999). ADAPTS: Adaptive Hypermedia for a Web-Based Performance Support System. Brusilovsky, P. and De Bra, P. (Eds.). *Proceedings of Second Workshop on Adaptive Systems and User Modelling on Word Wide Web Conference*, Toronto and Banff, Canada, pp. 23-24.

Brusilovsky, P. and Pesin, L. (1994). ISIS-Tutor: An adaptive hypertext learning environment. In: H. Ueno and V. Stefanuk (Eds.) *Proceedings Symposium on Knowledge-Based Software Engineering*, Pereslavl-Zalesski, Russia, EIC, pp. 83-87.

Brusilovsky, P. and Schwarz, E. (1997c). User as Student: Towards an Adaptive Interface for Advanced Web-based Applications. In Jameson, A., Paris, C. and Tasso, C. (Eds.), *Proceedings of 6th International Conference on User Modeling*, Chia, Springer-Verlag, pp. 177-188.

Brusilovsky, P., Schwarz, E. and Weber, G. (1996a). ELM-ART: An Intelligent Tutoring System on World Wide Web. Frasson, C., Gauthier, G. and Lesgold, A. (Eds.), *Proceedings of 3rd*

International Conference on Intelligent Tutoring Systems, (Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1086, pp. 261-269.

Bryman, A. and Cramer, D. (1997). *Quantitative Data Analysis*. London, Routledge.

Bunt, H.C. (1995). Dialogue Control Functions and Interaction Design. In Beun, R.J. Baker, M. Reiner, M. (Eds.), Dialogue and Instruction, Modeling Interaction in Intelligent Tutoring Systems. *Proceedings of the NATO Advanced Research Workshop on Natural Dialogue and Interactive Student Modeling*, Springer-Verlag, Berlin, pp. 197-214.

Burns, J. R. and Morgeson, J. D. (1988). An Object-Oriented World-View for Intelligent Discrete, Next-Event Simulation. *Management Science*, **34** (12), pp. 1425-1440.

Burns, H. L., and Capps, C. G. (1988). Foundations of intelligent tutoring systems: An introduction. In Polson, M. C., and Richardson, J.J. (Eds.), *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum, Hillsdale, NJ, pp. 1-19.

Burton, R.R. (1988). The Environment Module of Intelligent Tutoring Systems. Polson, M.C. and Richardson, J. J. (Eds.) *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum, Hillsdale, N.Y., pp. 109-142.

Burton, R. R., and Brown, J. S. (1982). An Investigation of Computer Coaching. Sleeman, D. H. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. Academic Press, New York, pp. 79-98.

Burton, R. R. and Brown, J. S. (1979). An Investigation of Computer Coaching for Informal Learning Activities. *International Journal of Man-Machine Studies*, 11, pp. 5-24.

Burton, R. R. (1982). Diagnosing bugs in a Simple Procedural Skill. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems*, Academic Press, London, pp. 157-183.

Burton, R. R., and Brown, J. S. (1982). An Investigation of Computer Coaching for Informal Learning Activities. Sleeman, D. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. Academic Press, London, pp. 79-98.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. (1996). *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley, West Sussex, UK.

Buss, A.H. and Stork, K.A. (1997). *Simkit User Manual.*

Buss, A.H. and Stork, K.A. (1996). Discrete Event Simulation on the World Wide Web Using Java. *Proceedings of the Winter Simulation Conference*. Morrice, D. and Charnes, J. (Eds.), Coronado, CA., pp. 780-785.

Bylander, T. and Chandrasekaran, B. (1988). Generic Tasks in Knowledge Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition. In Gaines, B. and Boose, J. (Eds.), *Knowledge Acquisition for Knowledge Based Systems*, Academic Press, London.1, pp. 65-77.

Byrne, M. D. and Bovair, S. (1997). A Working Memory Model of Common Procedural Error. *Cognitive science*, **21** (1), pp. 31-61.

Calvi, L. and De Bra, P. (1997). Using Dynamic Hypertext to Create Multi-purpose Textbooks. In Müldner, T., and Reeves, T. C. (Eds.), *Proceedings of World Conference on Educational Multimedia/Hypermedia and World Conference on Educational Telecommunications,* Calgary, Canada. AACE, Charlottesville, VA., pp. 130–135.

Canfield, A.M., Schwab, S., Merrill, M.D., Li, Z. and Jones, M.K. (1992). Instructional Transaction Theory: Resource Mediation. In Giardin, M. (Ed.) *Interactive Multimedia Learning Environment – Human factors and Technical Consideration on Design Issues*, Springer-Verlag, Heideberg, pp. 75-81.

Carbonaro, A., Maniezzo, V., Roccetti, M., and Salomoni, P. (1995). Modelling the Student in Pitagora 2.0. *User Modeling and User-Adapted Interaction* **4**(4), pp.233-251.

Carbonell, J. R. (1970). AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems*, **11** (4), pp. 190-202.

Carbonell, J. (1986) Derivational Analogy; A Theory of Reconstructive Problem Solving and Expertise Acquisition. Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (Eds.), *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Morgan Kaufmann, Los Altos, CA, pp. 371-392.

Carroll, J.M. (1992). The Nurnberg Funnel: *Designing Minimalist Instruction for Practical Computer Skill.* MIT Press, Cambridge, MA.

Carroll, J., Aaronson, A. Learning by Doing with Simulated Intelligent Help. (1988). *Communications of ACM,* **31** (9), pp. 1064-1079.

Castelfranchi, C. (1995). Garanties for Autonomy in Cognitive Agent Architecture. Wooldridge, M. and Jennings, N.R. (Eds.) *Intelligent Agents: Theories, Architectures and Languages. Artificial Intelligence*. Springer-Verlag, Berlin, 890, pp. 56-70.

Cawsey, A. (1992). *Explanation and Interaction*. Cambridge, MA, MIT Press.

Chan, T. W. (1996). Learning Companion Systems, Social Learning Systems, and the Global Social Learning Club. *Journal of Artificial Intelligence in Education*, **7** (2), pp. 125-159.

Chan, T., Chee, Y.S. and Lim, E.L. (1992). COGNITIO: An Extended Computational Theory of Cognition. In Frasson, C., Gauthier, G., McCalla, G.I. (Eds). *In Proceedings of the Second International Conference on Intelligent Tutoring Systems*, Springier-Verlag, Berlin, pp. 244-251.

Chandrasekaran, B. (1988). Generic Tasks as Building Blocks for Knowledge Based Systems: The Diagnosis and Routine Design Examples. *The Knowledge Engineering Review*, 3 (3), pp. 83-210.

Chandrasekaran, B. (1986). Generic Tasks for Knowledge-Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition, *IEEE Expert*, 1, pp.23-30.

Chandrasekaran, B., Josephson, J.R. and Benjamins, V.R. (1999). What are Ontologies, and Why do we need them? *IEEE Intelligent Systems*, pp. 20-26.

Chandrasekaran, B., Josephson, J.R. and Benjamins, V.R. (1998) The Ontology of Tasks and Methods. In Gaines, B.R. and Musen, M. (Eds.) *Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 6-21.

Chandrasekaran, B., Johnson, T.R, and Smith, J.W. (1992). Task-Structure Analysis for Knowledge Modelling, *Communications of the ACM*, 35, pp. 124-137.

Chi, M. T. H. (2000). Self-Explaining: The Dual Process of Generating Inferences and Repairing Mental Models. In Glaser, R. (Ed.). *Advances in Instructional Psychology*, Mahwah, NJ, Lawrence Erlbaum Associates, pp. 161-238.

Chi, M. T. H., De Leeuw, N.,Chiu, M.H. and LaVancher, C. (1994). Eliciting Self-Explanations Improves Understanding. *Cognitive Science*, 18, pp. 439-477.

Chipman, S.F. (1993). Gazing Once More into the Silicon Chip: Who's Revolutionary *Now?* Lajoie, S.P. and Derry, S.J. (Eds.). *Computers as Cognitive Tools*. Lawrence Erlbaum, Hillsdale, NW, pp. 341-367.

Chun, H. W. and Lai, E.M.K. (1997). Intelligent Critic System for Architectural Design. *IEEE Transactions on Knowledge and Data Engineering*, **9** (4), pp. 625-639.

Chu, R. W., Mitchell, C. M. and Jones, P. M. (1995). Using the Operator Function Model and OFMspert as the Basis for an Intelligent Tutoring System: Towards the Tutor/Aid Paradigm for Operators of Supervisory Control Systems. *IEEE Transactions on System, Man, and Cybernetics*, **25** (7), pp. 1054-1075.

Clancey, W. J. (1993). GUIDON-MANAGE Revisited: A Socio-Technical Systems Approach, *Journal of Artificial Intelligence in Education*, 4, pp. 5-34.

Clancey, W. J. (1992). Representation of Knowing: Defense of Cognitive Apprenticeship. *Journal of Artificial Intelligence in Education*, 3, pp. 139-168.

Clancey, W.J. (1991). The Knowledge Level Reinterpreted: Modeling how Systems Interact. Machine Learning, **4**(3/4), pp. 285-291.

Clancey W.J. (1990). Knowledge-Based Tutoring - The GUIDON Program. MIT Press Cambridge.

Clancey, W. (1987). Methodology for Building Intelligent Tutoring Systems. In Greg, P. Kearsley, (Eds.). *Artificial Intelligence and Instruction Applications and Methods*, Addison-Wesley, Reading, MA, pp. 193-227.

Clancey, W. J. (1986). Qualitative Student Models. *Annual Review of Computer Science*, 1, pp. 391 - 450.

Clancey, W. J. (1985). Heuristic Classification. *Artificial Intelligence*, **27** (3), pp. 289 - 350.

Clancey, W.J. (1983). The Epistemology of a Rule-Based Expert System a Framework for Explanation. *Artificial intelligence*, 20, pp. 215-225.

Clancey, W.J. (1982) Tutoring Rules for Guiding a Case Method Dialogue. In *Intelligent Tutoring Systems*, Academic Press, London, pp. 201-225.

Clancey, W. J. (1979). Tutoring Rules for Guiding a Case Method Dialogue. *International Journal of Man-Machine Studies*, **11** (9), pp. 25-49.

Clancey W.J., and Soloway E. (1991) *Artificial Intelligence and Learning Environments*. MIT Press, Reading, MA.

Clark, A. (1997). *Being There: Putting Body, Brain, and World Together Again*. MIT: MA, Cambridge Press.

Clibbon, K. (1995). Conceptually Adapted Hypertext for Learning. In Katz, I., Mack, R., and Marks, L. (Eds.), *Proceedings of Computer Human Interaction*. Denver, ACM. pp.224-225.

Coad, P., North, D. and Mayfield, M. (1995). *Object Models: Strategies, Patterns, and Applications*, Yourdon Press, Inglewood Cliffs, N.J.

Cofer, C. N. (1975). *The Structure of Human Memory*. Freeman, W. H. and Company. San Francisco, CA.

Collins, A. (1977). Processes in Acquiring Knowledge. Anderson, R. C. Spiro, R. J. and Montague, W. E. (Eds.), *Schooling and the Acquisition of Knowledge*. Lawrence Erlbaum, Hillsdale, NJ, pp. 339-363.

Collins, J. A., Greer, J. E., and Huang, S. X. (1996). Adaptive Assessment Using Granularity Hierarchies and Bayesian Nets. Frasson, C., Gauthier, G., and Lesgold, A., (Eds.), Proceedings of the Third *International Conference on Intelligent Tutoring Systems*. Springer-Verlag, Berlin, pp. 569-577.

Collins, A., Warnock, E. H. and Passafiume, J. J. (1975). Analysis and Synthesis of Tutorial Dialogues. In Bower, G. H. (Ed.), *The Psychology of Learning and Motivation*, Academic Press, 9, pp. 49-87.

Conati, C., Gertner, A., VanLehn, K., and Druzdzel, M., (1997). On-line Student Modeling for Coached Problem Solving Using Bayesian Networks. *Proceedings of Sixth International Conference on User Modelling*, pp. 231-242.

Conati, C. and VanLehn, K. (1996). POLA: A Student Modelling Framework for Probabilistic online Assessment of Problem Solving Performance. In Chin, D. N., Crosby, M., Carberry, S. and Zukerman, I. (Eds.), *Proceedings of the Fifth International Conference on User Modelling*, KailuaKona, User Modelling, Inc., Hawaii, pp. 75-82.

Constantine, L. L. and Lockwood, L. A. (1999). *Software for Use*. Addison-Wesley. Reading, MA.

Coolican, H. (1990). *Research Methods and Statistics in Psychology*. Hodder and Stoughton, London.

Corbett, A. T., Anderson, J. R., and O'Brien, A. T. (1995). Student Modelling in the ACT Programming Tutor. In Nichols, P. D., Chipman, S. F., and Brennan, R. L., (Eds.), *Cognitively Diagnostic Assessment*, Lawrence Erlbaum, Hillsdale, NJ., pp. 19-42.

Cortina, J.M. (1993). What is Coefficient Alpha? An Examination of Theory and Applications. *Journal of Applied Psychology*, **78** (1), pp. 98-104.

Cowan, N., Wood, N. L., and Borne, D. N. (1994). Reconfirmation of the Short-Term Storage Concept. *Psychological Science*, **5** (2), pp. 103-106.

Cox, B. J. (1996). *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, MA.

Cristina, C, Abigail S. Gertner, VanLehn, K, and Druzdzel, M. (1997). On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. Jameson, A., Paris, C., and Tasso, C. (Eds.). *Proceedings of Sixth International Conference on User Modelling*, Vienna, Springer-Verlag, New York, pp. 231-242.

Cronbach, L. J. (1990). *Essentials of Psychological Testing*. Harper and Row, New York.

Cronbach, L. J. and Snow R. E. (1977). Aptitudes and Instructional Methods: *A Handbook for Research on Interactions*. Irvington, New York.

Daganzo, C. (1996). *Logistics Systems Analysis*, Springer-Verlag.

Davis, R., Buchanan, B. and Shortliffe, E. (1985). Production Rules as a Representation for a Knowledge Based Consultation Program. In Brachman, R. J. and Levesque, H. J. (Eds.), *Readings in Knowledge Representation*, Los Altos, CA: Morgan Kaufmann Publishers, Inc. pp. 371-387.

Davies, R. and O'keefe, R. (1992). *Simulation Modelling with Pascal*. Prentice Hall. London. U.K.

Davydov, V.V. (1988). Learning Activity: The Main Problems Needing Further Research. *Activity Theory*, **1** (1-2), pp. 29-36.

deKleer, J. Doyle, J., Steele, G. L. Jr. and Sussman, G. J. (1985). AMORD: Explicit Control of Reasoning. In Brachman, R. J. and Levesque, H. J. (Eds.), *Readings on Knowledge Representation*. Los Altos, CA: Morgan Kaufmann Publishers, Inc. pp. 345-355.

DeMarco, T. (1982). *Controlling Software Projects: Management, Measurement and Estimation*. Yourdon Press, Inglewood Cliffs, N.J.

Dick, W. (1995). Instructional Design and Creativity: A Response to the Critics. *Educational Technology*, **35** (40), pp. 5-11.

Dick, W., and Carey, L.M. (1990). *The Systematic Design of Instruction*. Harper Collins, Glenview, IL.

Dickinson, D. (1995). Multimedia Myths. *Australian Personal Computer*, **16** (10), pp. 144-145.

Dickinson, D. (1995). Multimedia Myths. *Australian Personal Computer*, **16** (10), pp. 144-145.

Dieterich, H., Malinowski, U., Kühme, T. and SchneiderHufschmidt, M. (1993). State of the Art in Adaptive User Interfaces. In SchneiderHufschmidt, M., Kühme T. and Malinowski, U. (Eds.), *Adaptive User Interfaces: Principles and Practice*. Amsterdam, pp. 13-48.

Dillenbourg, P. and Self, J. (1992). A Framework for Learner Modelling. *Interactive Learning Environments*, **2** (2), pp.111-137.

diSessa, A. (1993). Toward an Epistemology of Physics. *Cognition and Instruction*, **10** (2-3), pp. 105-225.

Dixon, J. R. (1987). On Research Methodology Towards a Scientific Theory of Engineering Design, *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing*, Academic Press, **1** (3), pp. 145-157.

Doukidis, I. and Angelides, M. (1994). A Framework for Integration Artificial Intelligence and Simulation. *Artificial Intelligence Review*, 8, pp. 55-85.

Du Boulay, B., O'Shea, T. and Monk, J. (1981). The Black Box Inside the Glass Box. *International Journal of Man-Machine Studies*, 14, pp. 237-249.

Durkin, J. (1994). Knowledge Acquisition. Durkin, J. (Ed.), *Expert Systems: Design and Development*, pp. 518-599.

Eckstein, R., Loy, M., and Wood, D. (1998). *Java Swing*. O'Reilly and Associates. Sebastopol, CA.

Eisenstadt, M., Price, B. A., and Domingue, J. (1992). Software Visualisation: Redressing Intelligent Tutoring Systems Fallacies. *Proceedings of the NATO Advanced Research Workshop on Cognitive Models and Intelligent Environments for Learning Programming*, Margherita, S. (Ed.), Genova, Italy, pp. 123-135.

Ellis, H. C. and Hunt, R. R. (1993). Fundamentals of Cognitive Psychology. Fifth Edition. Brown and Benchmark/Brown, C., Publishers. Madison, WI.

Eliot, C., Neiman, D., and Lamar, M. (1997). Medtec: A Web-based Intelligent Tutor for Basic Anatomy. In: Lobodzinski S. and Tomek. I. (Eds.), *Proceedings of World Conference of the WWW, Internet and Intranet*, Toronto, Canada, AACE, Charlottesville, VA., pp. 161-165.

Eliott, C., Rickel, J. and Lester. J. (1999). Lifelike Pedagogical Agents and Affective Computing: An Exploratory Synthesis. In Wooldridge, M. and Veloso, M. (Eds.), *Artificial Intelligence Today, Lecture Notes in Computer Science* 1600, Springer-Verlag, pp. 195-212.

Elio, R. and Scharf, P.B. (1990). Modelling Novice-to-Expert Shifts in Problem Solving Strategy and Knowledge Representation. *Cognitive Science*, 14, pp. 579-639.

Eliot, C. and Woolf, B. (1995). *An Adaptive Student Centred Curriculum for and Intelligent Training System*, Woolf, B. (Ed.), Kluwer Academic Publishers, pp. 1-20.

Elsom-Cook, M. T. (1991). Dialogue and Teaching Styles. In Elson-Cook, M. (Ed.). Teaching Knowledge and Intelligence, Paul Chapman Publishing, London, pp. 61-84.

Elsom-Cook, M. (1990). Guided Discovery Tutoring. In Elson-Cook, M. (Ed.). *Guided Discovery Tutoring*, Paul Chapman Publishing, London, pp. 3-23.

Ely, D., Januszewski, A. and Le Blanc. G. (1988). *Trends and Issues in Educational Technology*. Syracuse University Press.

Engels, G., Lewerentz, C., Nagl, M., Schafer, W. and Schurr. A. (1992). Building Integrated Software Development Environments, Part 1: Tool Specification. *ACM Transactions on Software Engineering and Methodology*, **1** (2), pp.135-167.

Eriksson, H., Shahar, Y., Tu, S.W., Puerta, A.R., and Musen, M.A. (1996). Task Modelling with Reusable Problem-Solving Methods. *Artificial Intelligence*, **79** (2), pp. 293-326.

Eriksson, H., Puerta, R. and Musen, A. (1994). Generation of Knowledge-Acquisition Tools from Domain Ontologies. *International Journal of Human-Computer Studies*, 41, pp. 425-453.

Fensel, D., Motta, E., Decker, S., and Zdrahal, Z. (1997). Using Ontologies for Defining Tasks, Problem-Solving Methods and their Mappings. In Plaza, E. and Benjamins, V. R., (Eds.), *Knowledge Acquisition, Modeling and Management*, Springer-Verlag, Berlin, pp. 113-128.

Fensel, D. (1994). A Comparison of Languages which Operationalise and Formalise KADS Models of Expertise. *The Knowledge Engineering Review,* **9** (2), pp. 105-146.

Ferguson-Hessler, M. G. M., and Jong, T. d. (1990). Studying Physics Texts: Differences in Study Processes Between Good and Poor Solvers. *Cognition and Instruction*, 7, pp. 41-54.

Frakes, W.E, (1994). Success Factors of Systems Reuse. In *IEEE Software* **11** (5), pp.15-19.

Friedman, N., Goldszmidt, M., Heckerman, D., and Russell, S.J. (1997). Challenge: What is the Impact of Bayesian Networks on Learning? *International Journal of Computers Artificial Intelligence*, 1, pp. 10-15.

Fontatine, D., Le Beux P., Riou, C., Jacuelinet, C. (1994). An Intelligent Computer-Assisted Instructional system for Clinical Case Teaching. *Methods of Information in Medicine*, pp. 433-445.

Ford, K. M., Bradshaw, J. M., Adamswebber, J. R., and Agnew, N. M. (1993). Knowledge Acquisition as a Constructive Modelling Activity. *International Journal of Intelligent Systems*, **8** (1), pp. 9-32.

Fowler, N., Cross, S., and Owens, C. (1995). The ARPA: Rome Knowledge Based Planning Initiative. *IEEE Expert*, **10** (1), 4-9.

Fox, S. and Leake, D. (1994). Using Introspective Reasoning to Guide Index Refinement in Case-Based Reasoning. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum, Atlanta, GA, pp. 324-329.

Fox, B. A. (1993). *The Human Tutorial Dialogue Project: Issues in the Design of Instructional Systems*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Fu, M. C., Hayes, C. C., and East, E. W. (1997). SEDAR: Expert Critiquing System for Flat and Low-Slope Roof Design and Review. *Journal of Computing in Civil Engineering*, **11** (1), pp. 60-68.

Gaines, B.R. (1994). Class Library Implementation of an Open Architecture Knowledge Support System. *International Journal of Human-Computer Studies*, **41** (1/2), pp. 59-107.

Gagne, R.M. (1985). *The Conditions of Learning and the Theory of Instruction* CBS College Publishing, New York.

Gagne, R. M. and Briggs, L. J. (1979). Principles of Instructional Design. Secnd Edition, Rinehart and Winston, New York.

Gagne, R., Briggs, L. and Wenger, W. (1992). *Principles of Instructional Design.* Fourth Edition, HBJ College Publishers, Fort Worth, TX.

Gaines, B.R., Rappaport, A. and Shaw, M.L.G. (1992). Combining Paradigms in Knowledge Engineering. *Data and Knowledge Engineering*, 9, pp. 1-18.

Galliers, R. D. (1992). *Information Systems Research: Issues, Methods and Practical Guidelines,* Blackwell Scientific Publications, Oxford.

Galliers, R.D. (1991). Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy. In Nissen, H.E., Klein, H.K., and Hirschheim, R. (Eds.), *Information Systems Research: Contemporary Approaches and Emergent Traditions.* Holland Amsterdam, pp. 327-345.

Galliers, R. D. and F. F. Land (1987). Choosing an Appropriate Information Systems Research Methodology, *Communications of the ACM,* **30** (11), pp. 900-902.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns: Elements of *Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

Gardner, M., Greeno, J. G., Reif, F., Schoenfeld, A. H., diSessa, A., and Stage, E. (1990). *Toward the Scientific Practice of Science Education*. Lawrence Erlbaum, Hillsdale.

Garlan, D. and Shaw, M. (1993). An Introduction to Software Architecture. In Ambriola, V. and Tortora, G. (Eds.). *Advances in Software Engineering and Knowledge Engineering*, Singapore, 1993. World Scientific Publishing Company. pp. 1-39.

Gebhardt, E. (1978). A Critique of Methodology: Introduction. Aranto, A. and Gebhardt, E. (Eds), *The Essential Frankfurt School Reader*. Basil Blackwell, Oxford. pp. 371-406.

Geller, J., Perl, Y and Neuhold, E.J. (1991). Structural Schema Integration in Heterogeneous Multi-Database Systems Using the Dual Model. Kambayashi, Y. Rusinkiewicz, M. and Sheth, A. (Eds.), *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, IEEE Computer Society Press, pp. 200-203.

Gennari, J. H., Tu, S. W., Rothenfluh, T. E., and Musen, M. A. (1994). Mapping Domains to Methods in Support of Reuse. *International Journal of Human-Computer Studies*, 41, pp. 399-424.

Gentner, D.R. and Stevens, A.L. (1983). *Mental Models*. Lawrence Erlbaum, Hillsdale, NY.

Gertner, A. and VanLehn, K (2000). Andes: A Coached Problem Solving Environment for Physics. In Gauthier, G., Frasson, C. and VanLehn, K. (Eds.) Proceedings of 5th *International Conference on Intelligent Tutoring Systems,* Berlin: Springer, pp. 131-142.

Gertner A. S. and Webber B. L. (1998). TraumaTIQ: On-Line Decision Support for Trauma Management. *IEEE Intelligent Systems*, pp. 32-39.

Gibbs, G. (1995). Research into Student Learning. Research, Teaching and Learning. In Smith, B., and Brown, S. (Eds.). *Higher Education*. Kogan Page Ltd., London, pp. 17-29.

Ginsberg M.L. (1991). Knowledge Interchange Format: The KIF of Death. *Artificial Intelligence Magazine*, **12** (3), pp. 57–63.

Gitomer, D., Steinberg, H., S., L., and Mislevy, R. J. (1995). Diagnostic Assessment of Troubleshooting Skill in an Intelligent Tutoring System. In Nichols, P., Chipman, S., and Brennan, R., L., (Eds.), *Cognitively Diagnostic Assessment*. Erlbaum, Hillsdale, NJ., pp. 43-71.

Glaser, R. and Bassok, M. (1989). Learning Theory and the Study of Instruction. *Annual Review of Psychology,* 40:631-636.

Goldstein, I. P. (1979). The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. *International Journal of Man-Machine Studies*, 11, pp. 51-77.

Good, I. J. (1983). *Good Thinking*. University of Minnesota Press.

Goodkovsky, V.A, Kirjustin, E.V, Bulekov, A.A. (1994). Shell, Tool, and Technology for Pop Class ITS production. Brusilovsky, P., Dikareve, S., Greer J. and Pertrushin, V. (Eds). *Proceedings of East-West International Conference on Computer Technology in Education*. Crimea, Ukraine. Part 1, pp. 87-92.

Goodyear, P. (1991). *Teaching Knowledge and Intelligent Tutoring*. Ablex Publishing, Norwood, NJ.

Gosling, J., Joy, B., Steele, G., (2000). *The Java Language Specification*. Second Edition. Addison-Wesley, Reading, MA.

Greeno, J. G., & the Middle School Mathematics Through Applications Projects Group. (1998). The Situativity of Knowing, Learning, and Research. *American Psychologist*. **53** (1), pp. 5-26.

Gruber, T. (1995). *Ontolingua Overview*. http://www-sl.stanford.edu/kst/ontolingua.html.

Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, **5** (2), pp.199-220.

Goldstein, P.I. (1982). The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. Sleeman, D. H. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems*, London, Academic Press, pp. 51-77.

Gonschorek, M. and Herzog, C. (1995). Using Hypertext for an Adaptive Help System in an Intelligent Tutoring System. In Greer, J. (Eds.), *Proceedings of 7th World Conference on Artificial Intelligence in Education*, Washington, DC., AACE, Charlottesville, VA., pp. 274-281.

Graesser, A. C. and Clark, L. F. (1985). *Structures and Procedures of Implicit Knowledge,* Ablex Publishing, NJ.

Guarino, N., and P. Giaretta (1995). Ontologies and Knowledge Bases, Towards a Terminological Clarification. Mars, I. (Ed.), *Towards Very Large Knowledge Bases,* IOS Press, NJ, pp.25-32.

Guarino N. (1997). Understanding, Building and Using Ontologies. A Commentary to Using Explicit Ontologies in Knowledge Based Systems, by van Heijst, Schreiber, and Wielinga. *International Journal of Human and Computer Studies* 46 (2/3), pp. 293-310.

Guindon, R., Krasner, H., and Curtis, W. (1987). Breakdown and Processes During Early Activities of Software Design by Professionals. Olson, G. M. and Sheppard S., (Eds.). *Proceedings of Second Workshop on Empirical Studies of Programmers*, Ablex Publishing, Norwood, NJ, pp. 65-82.

Habermann, A. N. and Notkin. D. (1986). Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering*, **12** (12), pp. 1117-1127.

Haddawy, P., Kahn, C.E., and Butarbutar. M. (1994). A Bayesian Network Model for Radiological Diagnosis and Procedure Selection: Workshop of Suspected Gallbladder Disease. *Medical Physics*, 21, pp. 1185 - 1192.

Halff, H. M. (1988). Curriculum and Instruction in Automated Tutors. Polson, M. C. and Richardson, J. J. (Eds.). *Foundations of Intelligent Tutoring Systems*, Lawrence Erlbaum, Hillsdale, NJ, pp. 79-108.

Hammond, K. (1989). *Case-Based Planning*. Academic Press.

Harris, B. and Cook, D. (1998). Integrating Hierarchical and Analogical Planning. Cook, J. C. (Ed.), *Proceedings of the Eleventh International Florida Artificial Intelligence Research Symposium*, Sanibel Island, FL, pp. 126-130.

Hartley, J. R., and Sleeman, D. H. (1973). Towards More Intelligent Teaching Systems. *International Journal of Man-Machine Studies*, 2, pp. 215-236.

Hayes-Roth, B. (1985). A Blackboard Architecture for Control. *Artificial Intelligence*, 26, pp. 251-321.

Hayes-Roth, B., and Doyle, P. (1998). Animate Characters. *Autonomous Agents and Multi-Agent Systems*, **1** (2), pp.195-230.

Hayes-Roth, F., Waterman, D. A., and Lenat, D.B. (1983). *Building Expert Systems*. Reading, MA, Addison-Wesley.

Hawkes, L. W., Derry, S. J., and Rundensteiner, E. A. (1990). Individualised Tutoring Using an intelligent Fuzzy Temporal Relational Database. *Journal of Artificial Intelligence in Education*, 1, pp. 43-56.

Heckerman. D. (1997). Bayesian Networks for Data Mining. *Data Mining and Knowledge Discovery*, **1** (1), pp. 79-119.

Heckerman, D., Horvitz, E. and Nathwani, B. (1992). Toward Normative Expert Systems: Part I. The Pathfinder Project. *Methods of Information in Medicine*, 31, pp. 90-105.

Heise, D.R. (1969). Methodology of the Semantics Differential. *Psychological Bulletin,* 72, pp. 93-95.

Hinrichs, T. (1992). *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Hlupic, V. (1993*). Simulation Modelling Software Approaches to Manufacturing Problems.* London School of Economics and Political Science, University of London. Unpublished Ph.D. Thesis.

Hlupic, V., Paul, R.J. and Z, Irani. (1999). Evaluation Framework for Simulation Software. *International Journal of Advanced Manufacturing Technology,* 15, pp. 336-382.

Hodson, D. (1988). Towards a Philosophically More Valid Science Curriculum. *Science Education*, **72** (1), pp. 19-40.

Hoffman, R.R. (1987). The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology. *Artificial Intelligence Magazine*, **8** (2), pp. 53-67.

Hohl, H., Böcker, H., and Gunzenhäuser, R. (1996). Hypadapter: An Adaptive Hypertext System for Exploratory Learning and Programming. *User Modelling and user Adapted Interaction*, **6** (2-3), pp. 131-156.

Hollan, J., Hutchins, E., and Kirsh, D. (2000). Distributed Cognition: Towards a New Foundation for Human-Computer Interaction Research. ACM *Transactions of Computer-Human Interaction*, **7**(2), pp. 174-196.

Horvitz, E. (1999). Uncertainty, Action, and Interaction: In Pursuit of Mixed-Initiative Computing, *Intelligent Systems*, IEEE Computer Society, pp. 17-20.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, R. (1998). The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. Cooper, G.F. and Moral, S. (Eds), *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Francisco, pp. 256-265.

Howell, F.W. (1997). *The Simjava User Manual.* University of Edinburg.

Hume, G., Michael, J., Rovick, A., and Evens, M. (1996). Hinting as a Tactic in One-on-One Tutoring. *The Journal of Learning Sciences*, Lawrence Erlbaum, Hillsdale, NJ., **5** (1), pp. 23-47.

Hutchins, E. L. (1995). *Cognition in the Wild.* Cambridge, MA: The MIT Press.

Ikeda, M., Go, S., and Mizoguchi, R. (1997). Opportunistic group formation. In Boulay, B. d. and Mizoguchi, R. (Eds.) *Proceedings of 8th World on Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, Kobe, Japan, Amsterdam: IOS, pp. 167-174.

Ikeda, M and R. Mizoguchi. (1994). FITS: A Framework for ITS - A Computational Model of Tutoring, *Journal of Artifical Intelligence in Education*, **5** (3), pp.319-348.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison Wesley, Reading, Mass.

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. (1992). *Object-Oriented Software Engineering - A Use Case Driven Approac*h. ACM Press.

Jacobson, I, Griss, M. and Jonsson, P. (1997). *Software Reuse*, Addison Wesley, Reading, MA.

Jameson, A. (1996). Numerical Uncertainty Management in User and Student Modelling: An Overview of Systems and Issues. *User Modelling and User-Adapted Interactio*n, 5, pp. 193-251.

Jonassen, D., Shute, V. J., Willis, R. E., and Torreano, L. A. (1999). Decompose, Network, Assess (DNA). In Jonassen, D. H., Tessmer, M. and Hannum, W. H. (Eds.*), Task Analysis Methods for Instructional Design*, Lawrence Erlbaum, Mahwah, NJ, pp. 131-138.

Jansson, D. and Smith, S. M. (1991). Design Fixation. *Design Studies*, 12, pp. 3-11.

Johnson, W.L., Rickel, J.W. and Lester. J.C. (2000). Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education*, 11, pp. 47-78.

Johnson, W. L., Rickel, J., Stiles, R., and Munro, A. (1998). Integrating Pedagogical Agents into Virtual Environments. Presence, *Teleoperators and Virtual Environments* **7** (6), pp.523-546.

Johnson, W. L., and Rickel, J. (1998). Steve: An Animated Pedagogical Agent for Procedural Training in Virtual Environments. *SIGART Bulletin*, 8, pp. 16-21.

Jonassen, D. (1988). *Instructional Designs for Microcomputer Courseware*Lawrence Erlbaum, Hillsdale, NJ.

Kantner, L. and Rosenbaum, S. (1997). Usability Testing of WWW Sites: Heuristic Evaluation vs. Laboratory Testing. *In Proceeding of the Fiventh Annual International Conference on Computer Documentation*, Association for Computing Machinery. New York, pp. 153-160.

Katz, S., and Lesgold, A. (1993). The Role of the Tutor in Computer-Based Collaborative Learning Situations. Lajoie, S. P., Derry, S. J. (Eds.), *Computers as Cognitive Tools*, Lawrence Erlbaum, Hillsdale, NY, pp. 289-318.

Katz, S., Lesgold, A., Eggan, G., and Gordin, M. (1993). Modelling the Student in SHERLOCK II. *Journal of Artificial Intelligence and Education* (Special Issue on Student Modeling), **3** (4), pp. 495-518.

Kaye, A. R. (1991). *Collaborative Learning through Computer Conferencing*. Springer-Verlag, Berlin.

Kambouri, M., Koppen, M., Villano, M. and Falmagne, J. C. (1994). Knowledge Assessment: Tapping Human Expertise by the QUERY Routine. *International Journal of Human-Computer Studies*, 40, pp. 119-151.

Kerninghan, B.W. and Lesk, M.E. (1979). *LEARN: Computer Aided Instruction on Unix*. (Second Edition). Bell Laboratories.

Kintsch, W. and Greeno, J. G. (1995). Understanding and Solving Word Arithmetic Problems. *Psychological Review*. 92, pp. 109-129.

Klein, H.K., Hirschheim, R., and Nissen, H.E. (1991). A Pluralist Perspective of the Information Systems Research Arena. In *Information Systems Research: Contemporary Approaches and Emergent Traditions*. Nissen, H.E., Klein, H.K., and Hirschheim, R. (Eds.) Holland Amsterdam, pp. 1-20.

Klemm, W. R. (1994). Using a Formal Collaborative Learning Paradigm for Veterinary Medical Education. *Journal of Veterinary Medicine Education*. 21, pp. 2-6.

Kline, P. (1993). *The Handbook of Psychological Testing*. London, Rouledge.

Klinker G, Bhola C, Dallemagne G, Marques D, McDermott J. (1991). Usable and Reusable Programming Constructs. *Knowledge Acquisition*, **3** (3), pp. 117-135.

Kirsh, D. (1996). Adapting the Environment Instead of Oneself. *Adaptive Behavior, 4*, 415-452.

Kintsch, W. and Greeno, J. G. (1995). Understanding and Solving Word Arithmetic Problems. *Psychological Review*. 92, pp. 109-129.

Koedinger, K., and Anderson, J. (1995). Intelligent Tutoring Goes to the Big City. In Greer. J. (Ed). *Proceedings of the International Conference on Artificial Intelligence in Education*, AACE: Charlottesville, VA., pp. 421-428.

Koedinger, K., Anderson, J.R., Hadley, W.H., and Mark, M. A. (1997). Intelligent Tutoring goes to School in the Big City. *International Journal of Artificial Intelligence in Education*, 8, pp. 30-43.

Koedinger, K. R., Anderson, J.R., Hadley, W.H., and Mark, M. A. (1995). Intelligent Tutoring Goes to School in the Big City. In Greer, J. (Ed.), *Proceedings of the 7th World Conference on Artificial Intelligence and Education*, AACE, Charlottesville, VA., pp. 421-428.

Koedinger, K., Suthers, D., and Forbus, K. (1999). Component-Based Construction of a Science Learning Space. *International Journal of Artificial Intelligence in Education*, 10, pp. 292-313.

Koedinger, K., Shuthers, D., and Forbus, K. (1998). Component Based Construction of a Science Learning Space. Goettl, B., Halff, H., Redfield, C., and Shute, V., (Eds.), *Fourth International Conference on Intelligent Tutoring Systems,* San Antonio, Texas, Springer-Verlag, Berlin, pp. 166-175.

Koen, B.V. (1985). *The Definition of the Engineering Method.* American Society of Engineering Education, Washington, D.C.

Kolodner, J. L. (1993). *Case-Based Reasoning*. Morgan Kaufamann, San Mateo, CA.

Kolodner, J. (1983). Maintaining Organisation in a Dynamic Long-Term Memory. *Cognitive Science*, 7, pp. 243-280.

Koton, P. (1989). Smartplan: A Case-Based Resource Allocation and Scheduling System. In Hammond, K. (Ed.), *Proceedings of the Case-Based Reasoning Workshop*, Morgan Kaufmann, San Mateo, DARPA, pp. 290-294.

Krasner, G. E. and Pope, S. T. A. (1988). A Cookbook for Using the Model ViewController User Interface Paradigm in Smalltalk80. *Journal of Object Oriented Programming*, **1** (3), pp. 26-49.

Kruchten, P. B. (1995). The 4+1 View Model of Architecture. *IEEE Software*, pp. 42-50.

Krueger, C. W. (1992). Software Reuse. *ACM Computing Surveys*, 24, pp. 131-183.

Kurland, D. M. and Pea, R. D. (1985). Children's Mental Models of Recursive LOGO Programming. *Journal of Educational Computing Research*, **1** (2), pp. 235-243.

Laird, D. (1985). Approaches to Training and Development. Addison- Wesley, Reading, MA.

Laird, J.E., Newell, A., and Rosenbloom, P.S. (1987). Soar: An Architecture for General Intelligence. *Artificial Intelligence*, **33** (1), pp.1-64.

Lajoie, S. P., Lesgold, A. M. (1992). Dynamic Assessment of Proficiency for Solving Procedural Knowledge Tasks. *Educational Psychologist*, **27** (3), pp. 365-384.

Langley, P., Wogulis, J. and Ohlsson, S. (1990). Rules and Principles in Cognitive Diagnosis. Frederiksen, N., Glaser, R., Lesgold, A. and Shafto, M. (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*, Lawrence Erlbaum, Hillsdale, NJ, pp. 217-250.

Latham, G.P. and Saari, L.M. (1979). Application of Social-Learning Theory to Training Supervisors through Behavioural Modelling. *Journal of Applied Psychology*, 64, pp. 239-246.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society*, 50, pp. 157-224.

Law, A.N., and Kelton, W.D. (2000). *Simulation Modeling and Analysis*. (Third Edition). McGraw Hill, New York.

Lazonder, A.W., and Van der Meij, H. (1995). Error-information in Tutorial Documentation: Supporting Users' Errors to Facilitate Initial Skill Learning. *International Journal of Human-Computer Studies*, 42, pp. 185 - 206.

LeFevre, J. and Dixon, P. (1986). Do Written Instructions Need Examples? *Cognition and Instruction*, 3, pp. 1-30.

Legree, P.J., Gillis, P.D. and Orey, M. (1993). The Quantitative Evaluation of Intelligent Tutoring System Applications: Product and Process Criteria. *Journal of Artificial Intelligence in Education,* **4** (2/3), pp. 209-226.

Lenat, D. B. (1995). CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM* , **38** (11), pp. 33-43.

Lesgold, A. (1988). Towards a Theory of Curriculum for Use in Designing Intelligent Instructional Systems. In Mandl, H. and Lesgold, A. (Eds.), *Learning Issues for Intelligent Tutoring Systems*. Springer-Verlag, New York, pp. 114-137.

Lesgold, A. M., Lajoie, S. P., Bunzo, M., and Eggan, G. (1993). SHERLOCK: A Coached Practice Environment for an Electronics Troubleshooting Job. Larkin, J. Chabay, R. and Scheftic, C. (Eds.). *Computer Assisted Instruction and Intelligent Tutoring systems: Establishing communication and Collaboration*, Lawrence Erlbaum, Hillsdale, NJ, pp. 201-238.

Lesgold, A., IvillFriel, J. and Bonar, J. (1989). Toward Intelligent Tutoring Systems for Testing. In Resnick, L. B. (Ed.). *Knowing, Learning and Instruction: Essays in Honor of Robert Glaser*. Lawrence Erlbaum, Hillsdale, NJ, pp. 337-360.

Lester, J. C., Stone, B. A., and Stelling, G. D. (1999). Lifelike Pedagogical Agents for Mixed-Initiative Problem Solving in Constructivist Learning Environments. *User Modelling and User-Adapted Interaction* 9, pp.1-44.

Lewis, R. L. (1996). Interference in Short-Term Memory: The Magical Number Two (or Three) in Sentence Processing. *The Journal of Psycholinguistic Research* 25, pp.93-115.

Liebenau, J. and Backhouse, J. (1990). *Understanding Information: An Introduction*. Macmillan.

Likert, R. (1932). *A Technique for the Measurement of Attitudes*. Colombia University Press, New York.

Little, M.C. (1997). http://cxxsim.ncl.ac.uk.

Livingstone, C., and Borko, N. (1989). Expert-Novice Differences in Teaching: A Cognitive Analysis and Implications for Teachers Education. *Journal of Teacher Education*, **40** (4), pp.36-42.

Lopez, B., and Plaza, E. (1993). Case-Based Planning for Medical Diagnosis, Komorowski, J., Ras, Z. W. (Eds.) *Methodologies for Intelligent Systems: Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, 689, pp. 96-105.

Luckham, D. C., and Vera, J. (1995). An Event Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, **21** (9), pp. 717-734.

Macmillan, S. A., Emme, D. and Berkowitz, M. (1988). Instructional Planners: Lessons learned. In Psotka, J., Massey, L. D. and Mutter, S. A. (Eds.), *Intelligent Tutoring Systems: Lessons learned*, Lawrence Erlbaum, Hillsdale, NJ, pp. 229-256.

Macromedia Inc. (1997). Authorware at http://www.macromedia.com/index.html.

Macrelle, M., Desmoulins, C., (1998). Macro Definitions, a Basic Component for Interoperability between ILEs at the Knowledge Level: Application to Geometry. Goettl, B., Halff, H., Redfield, C., and Shute, V., (Eds.), *Fourth International Conference on Intelligent Tutoring Systems*, San Antonio, Texas. Springer Verlag, Berlin, pp. 46-55.

Major, N. (1995). Modelling Teaching Strategies. *International Journal of Artificial Intelligence in Education*, **6** (2-3), pp. 117-152.

Major, N., Ainsworth, S. and Wood, D. (1997). REDEEM: Exploiting Symbiosis Between Psychology and Authoring Environments. *International Journal of Artificial Intelligence in Education*, **8** (3-4), pp. 317-340.

Major, N. and Reichgelt, H. (1991). Using COCA to Build an Intelligent Tutoring System in Simple Algebra. *Intelligent Tutoring Media*, **2** (3-4), pp.159-169.

Margolis, A.A. (1993). Vygotskian Ideas in Computer Assisted Instruction (CAI). *Activity Theory*, 13/14, 9-13.

Mark, M. A. and Greer, J. E. (1995). The VCR Tutor: Effective Instruction for Device Operation. *The Journal of the Learning Sciences*, **4** (2) pp. 209-246.

Mark, M. A. and Greer, J. E. (1993). Evaluation Methodologies for Intelligent Tutoring Systems, *Journal of Artificial Intelligence in Education*, 4, pp. 129-153.

Martin, J., and VanLehn, K. (1995). Student Assessment Using Bayesian Networks. *International Journal of Human-Computer Studies*, 42, pp. 575-591.

Martin, J. and VanLehn, K. (1993). OLAE: Progress Toward a Multiactivity, Bayesian Student Modeller. In Brna, S. P., Ohlsson, S. and Pain, H. (Eds.), *Proceedings of the World Conference on Artificial Intelligence in Education*, Association for the Advancement of Computing in Education, Charlottesville, VA, pp. 426-432.

Mayer, R. (1981). The Psychology of How Novices Learn Computer Programming. *Computing Surveys*, **13** (1), pp. 121-141.

McGraw, K.L. and Harbison-Briggs, K. (1989). *Knowledge Acquisition: Principles and Guidelines*. Prentice Hall, Englewood Cliffs, NJ.

McKendree, J. (1990). Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction*, 5, pp. 381-413.

McDermott, J. (1988). Preliminary Steps Toward a Taxonomy of Problem Solving Methods. Marcus, S. (Ed). *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic, Boston, pp. 225-256.

McLellan, H. (1992). Hyper Stories: Some Guidelines for Instructional Designers. *Journal of Research on Computing in Education*, **25** (1), pp. 28-48.

Merrill, D. (1996). Instructional Transaction Theory: Instructional Design Based on Knowledge Objects. *Educational Technology*, pp. 30-37.

Merrill, M. D. (1985). Where is the Authoring in Authoring Systems? *Journal of Computer-Based Instruction*, 12, pp. 90-96.

Merrill, M. D. (1983). Component Display Theory. In Reigeluth, C. M. (Ed.), Instructional Design Theories and Models: An Overview of their Current Status, Lawrence Erlbaum, Hillsdale, NJ, pp. 279-333.

Merrill, D. C., Reiser, B. J., Ranney, M. and Trafton, J. G. (1992). Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems. *Journal of the Learning Sciences*, 2, pp. 277-305.

Microsoft (2000). Microsoft Office 2000. *Redmond, Washington*. Microsoft Press, Richmond.

Microsoft (1998). Text-To-Speech Engine. *User Documentation*. Microsoft Press, Richmond.

Miller, G. A. (1965). The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *Psychological Review*, **63** (2), pp. 81-97.

Minton, S. (1990). Quantitative Results Concerning the Utility of Explanation Based Learning. *Artificial Intelligence*, 42, pp.363-391.

Mitchell, T.M., Caruana, R.,. Freitag, D., McDermott, J. and Zabowski, D. (1994). Experience with a Learning Personal Assistant, *Communications of the ACM*, **37** (7), pp. 81-91.

Mitchell, T. M., Mabadevan, S., and Louis I. Steinberg. (1990). LEAP: A Learning Apprentice for VLSI Design. In *Machine Learning: An Artificial Intelligence Approach*, Volume III. Morgan Kaufmann, San Mateo, CA., pp. 271-289.

Mize, J. H., Bhuskute, H. C., Pratt, D. B. and Kamath, M. (1992). Modelling of Integrated Manufacturing Systems Using an Object-Oriented Approach. *IEEE Transactions,* **24** (3), pp. 14-26.

Mizoguchi, R. (1993). Knowledge Acquisition and Ontology. In *Proceeding of Knowledge Based and Knowledge Systems Conference*, Tokyo, pp.121-128.

Mizoguchi, R., Sinitsa, K., and Ikeda, M. (1996). Task Ontology Design for Intelligent Educational/Training Systems. Frasson, S., Gauthier, G., and Lesgold, A. (Eds.), In the Proceedings of Third *International Conference on Intelligent Tutoring Systems,* Montreal, pp.12-14,

Mizoguchi, R., Tijerino, Y., M. Ikeda, M. (1995) Task Analysis Interview Based on Task Ontology. *Journal of Expert Systems with Applications*, **9**, (1), pp.15-25.

Möbus, C., Schröder, O. and Thole, H.J. (1993). A model of the Acquisition and Improvement of Domain Knowledge for Functional Programming. *Journal of Artificial Intelligence in Education*, **3** (4), pp. 449-476.

Monarchi, D. E., Puhr, G. I. (1992). A Research Typology for Object-Oriented Analysis and Design. *Communications of the ACM*, **35** (9), pp. 35-47.

Moore, J. D. (1996). Discourse generation for instructional applications: Making computer-based tutors more like humans. *Journal of Artificial Intelligence in Education*, **7**(2), 181-124.

Moore, J. D., Lemaire, B., and Rosenblum, J. A. (1996). Discourse Generation for Instructional Applications: Identifying and Exploiting Relevant Prior Explanations. *Journal of the Learning Sciences*, **5** (1), pp.49-94.

Mosier, J. and Smith, S. (1986). Applications of Guidelines for Designing User Interface Software. *Behaviour and Information Technology*, **5** (1), pp. 39-46.

Murray, T. (1999). Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art. *International Journal of Artificial Intelligence in Education,* **10** (1), pp. 98-129.

Murray, T. (1998). Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. *Journal of the Learning Sciences*, **7**, (1), pp. 5-64.

Murray, T. (1997). Expanding the Knowledge Acquisition Bottleneck for Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education*, **8** (3-4), pp. 222-232.

Murray, T. (1996). Having it All, Maybe: Design Trade-offs in ITS Authoring Tools. Frasson, C., Gauthier, G., and Lesgold, A., (Eds.), *Proceedings of the Third International Conference on Intelligent Tutoring Systems*, Springer-Verlag, Berlin, pp. 93-101.

Murray, T. (1993). Formative Qualitative Evaluation for "Exploratory" ITS Research. *International Journal of Artificial Intelligence in Education,* **4** (2-3), pp. 179-207.

Murray, T., Condit, C., Piemonte, J. Shen, T., and Kahn, Samia (1999). MetaLinks-A Framework and Authoring Tools for Adaptive Hypermedia. In Lajoie, S. (Ed.), Proceedings of the *9th World Conference on Artificial Intelligence in Education*, IOS Press, Amsterdam, pp. 744-746.

Murray, T. and Woolf, B. P. (1992). Tools for Teacher Participation in ITS Design. Frasson, C., Gauthier, G. and McCalla, G.I. (Eds.), *Intelligent Tutoring Systems*. Springer-Verlag, Berlin, pp. 593-600.

Musen, M. A. (1999). Scalable Software Architectures for Decision Support. *Methods of Information in Medicine*, 38, pp. 229-238.

Musen. M. (1998). Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. In Chute, C.G. (Ed.), *AMIA Annual Symposium*, Orlando, FL, pp. 46-52.

Musen. M.A. (1993). An Overview of Knowledge Acquisition. In David, J. M. Krivine, J. P. and Simmons, R. (Eds.). *Second Generation Expert Systems*, Berlin, Springer-Verlag, pp.405-427.

Musen, M.A. (1992). Dimensions of Knowledge Sharing and Reuse. *Computers and Biomedical Research*, 25 pp. 435-467.

Musen, M.A. (1989). Automated Support for Building and Extending Expert Models, *Machine Learning,* 4, pp. 349-377.

Musen, M., Gennari, J. H., Eriksson, H., Tu, S. W. and Puerta, A. R. (1995). PROTEGE II: Computer Support For Development Of Intelligent Systems From Libraries Of Components. *Proceedings of The Eighth World Congress on Medical Informatics*, Vancouver, B.C., Canada, 766-770.

Musen, M. and Schreiber, A. T. (1995). Architectures for Intelligent Systems Based on Reusable Components. *Artificial Intelligence in Medicine* 7, pp. 189-199.

Nakabayashi, K., Maruyama, M., Kato, Y., Touhei, H., and Fukuhara, Y. (1997). Architecture of an Intelligent Tutoring System on the WWW. In Boulay B. d. and Mizoguchi, R. (Eds.). *Proceedings of World Conference on Artificial Intelligence in Education on Artificial Intelligence in Education: Knowledge and Media in Learning Systems* Kobe, Japan, Amsterdam, IOS, pp. 39-46.

National Centre for Supercomputing Applications (NCSA). (1997). *The Common Gateway Interface*. http://hoohoo.ncsa.uiuc.edu/cgi/

Neighbors, J. M. (1992). The Evolution from Software Components to Domain Analysis, *International Journal of Software Engineering and Knowledge Engineering*, **2** (3), pp. 325-354.

Neighbors, J. M. (1984). The Draco Approach to Constructing Software from Reusable Components, *IEEE Transactions on Software Engineering* 10, pp. 564-574.

Nelson, M. (1995). *C++ Programmer's Guide to the Standard Template Library*. IDG Books Worldwide Inc., Foster City, CA.

Newell, A. (1990). *Unified Theory of Cognition*. Harvard University Press.

Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, 18, pp.87-127.

Newman, W. M. and Lamming, M. G. (1995). *Interactive System Design*. Addison-Wesley, Reading, MA.

Nielsen, J. (1994). Heuristic Evaluation. Nielsen, J., and Mack, R. L. (Eds.), *Usability Inspection Methods*. John Wiley, New York, pp. 25-64.

Nielson, J. (1993). *Usability Engineering*. Academic Press, Boston, MA.

Nielsen, J. and Mack, R.L. (1994). Usability Inspection Methods, John Wiley, New York, NY, pp. 49.

Nkambou, R. and Gauthier, G. (1996). Use of WWW Resources by an Intelligent Tutoring System. In *Educational Multimedia and Hypermedia*, pp. 527-532.

Noma, T., and Badler, N. I. (1997). A Virtual Human Presenter. *Proceedings of the International Journal of Computer in Artificial Intelligence Workshop on Animated Interface Agents: Making Them Intelligent*, pp.45-51.

Norman, D. (1993). Cognition in the Head and in the World: An Introduction to the Special Issue on Situated Action. *Cognitive Science*, **17** (1), pp. 1-6.

Norman, D. and Draper, S. (1987). *User Centred System Design*, Lawrence Erlbaum, Hillsdale, NJ.

Object Management Group (OMG). (1999). UML Specification V1.3: *Object Management Group Document*. http://www.omg.org.

Object Management Group (OMG). (1998). XML Metadata Interchange (XMI): *Object Management Group Document*. http://www.omg.org.

Object Management Group. (1997). UML Semantics. *Object Management Group Document*. http://www.omg.org.

O'Keefe, R. (1986). Simulation and Expert Systems: A Taxonomy and Some Examples. *Simulation*, **46** (1), pp. 10-16.

Ohlsson, S. (1994). Constrained-Based Student Modelling. In Greer, E.J. and McCalla, G.I. (Eds.). *Student Modelling: The Key to Individualised Knowledge-Based Instruction*, NATO ASI Series F: Computer and Systems Sciences, Special Programme: Advanced Educational Technology, Springer, Berlin, 125, pp. 167-189.

Ohlsson, S. (1993). Impact of Cognitive Theory on the Practice of Courseware Authoring, *Journal of Computer Assisted Learning*, **9** (4), pp. 194-221.

Ohlsson, S. (1992). Constraint Based Student Modelling, *Artificial Intelligence in Education*, **3** (4), pp. 429-447.

Ohlsson, S. (1991). Knowledge Requirements for Teaching: The Case of Fractions. In Goodyear, P. (Ed.), *Teaching Knowledge and Intelligent Tutoring*. Ablex Publishing, Norwood, NJ, pp. 25-59.

Ohlsson, S. (1987). Some Principles of Intelligent Tutoring. Lawler, R.W., and Yazdani, M., (Eds.), *Artificial Intelligence and Education, Vol. 1, Learning Environments and Tutoring Systems. Norwood*, Ablex Publishing, New York, pp. 203- 237.

Ohlsson, S. (1986). Some Principles of Intelligent Tutoring, *Instructional Science, 14*, pp. 293-326.

Okazaki, Y., Watanabe, K., and Kondo, H. (1996). An Implementation of an Intelligent Tutoring System (ITS) on the World-Wide Web (WWW). *Educational Technology Research* **19** (1), pp. 35-44.

Okazaki, Y., Watanabe, K., and Kondo, H. (1997). An Implementation of the WWW Based ITS for Guiding Differential Calculations. In Brusilovsky, P., Nakabayashi, K. and Ritter, S. (Eds.), *Proceedings of World Conference on Artificial Intelligence in Education Workshop on Intelligent Educational Systems on the World Wide Web*, 8th Kobe, Japan, ISIR, pp. 18-25.

Orfali, R., Harkey, D. and Edwards, J. (1996). *The Essential Distributed Objects: Survival Guide*. John Wiley, New York.

Orey, M.A., Trent, A., and Young, J. (1993). Development Efficiency and Effectiveness of Alternative Platforms for Intelligent Tutoring. Brna, P., Ohlsson, S., and Pain, H., (Eds.), *Proceedings of the World Conference on Artificial Intelligence in Education*, Association for the Advancement of Computing in Education, Charlottesville, VA, pp. 42-49.

O'Shea, T. (1982). A Self-Improving Quadratic Tutor. Sleeman, D and Brown, J.S. (Eds.). *International Tutoring Systems*, Academic Press, Boston, MA, pp. 309-336.

O'Shea, T and Self, J. (1983). *Learning and Teaching with Computers-Artificial Intelligence in Education*. Prentice Hall, Englewood Cliffs, NJ.

Osgood, C. E., Suci, G. J., and Tannenbaum, P. H. (1957). *The Measurement of Meaning*. University of Illinois Press, Urbana, IL.

Ostertag, E., Hendler, J., PrietoDiaz, R., and Braun, C. (1992). Computing Similarity in a Reuse Library System: An Artificial Intelligence Based Approach. *ACM Transactions on Software Engineering and Methodology*, **1** (3), pp.205-228.

Park, O., Perez, R. and Seidel, R. (1987). Intelligent CAI: Old Wine in New Bottles, or a New Vintage? In Kearsley, G. (Ed.) *Artificial Instruction and Instruction Applications and Methods*, Addison Wesley, Reading. MA, pp. 11-46.

Patrick, J. (1992). *Training: Practice and Research.* Academic Press, London.

Patton, M. Q. (1990). Qualitative Evaluation and Research Methods, *SAGE*, Publications, Inc.

Paul, R. and Balmer, D. (1993). *Simulation Modelling*. Chartwell Bratt. London, U.K.

Paul, R. and Hlupic, V. (1994). Designing and Managing a Degree Course in Simulation Modelling. *Proceedings of the Winter Simulation Conference*, Tew, J.D. Manivannan, S., Sadowski, D.A. and Seila, A.F. (Eds.), ACM Press, Orlando, pp. 1393-1398.

Paul, R.J., Taylor, S.J.E., Hlupic, V., and Baldwin, L.P. (1998). A Methodology for the Integration of Computer-Based Training into Simulation Modelling Courses. *Proceedings of European Simulation Conference*, Bargiela, A and Kerckhoffs, E. (Eds.), Nottingham, pp. 709-714.

Pea, R. D., (1993a) Learning Scientific Concepts Through Material and Social Activities: Conversational Analysis Meets Conceptual Change. *Educational Psychologist*, **28** (3), pp. 265-277.

Pea, R. D. (1993b). Distributed Multimedia Learning Environments: The Collaborative Visualisation Project, *Communications of the ACM*, 36, pp. 60-63.

Pearl, J. (1993). Belief Networks Revisited. *Artificial Intelligence*, 59, pp. 49-56.

Pearl, J. (1988a). On Probability Intervals. *International Journal of Approximate Reasoning*, 2, pp. 211-216.

Pearl, J. (1988b). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, (Second Edition), Morgan Kaufman, San Mateo.

Pennington, N. (1987). Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychology*, 19, pp. 295-341.

Perez, A.G., Benjamins, R.V. (1999). Applications of Ontologies and Problem-Solving Methods. In *AI Magazine*, **20** (1), pp.119-122.

Perez, T., Gutiérrez, J. and Lopistéguy, P. (1995). An Adaptive Hypermedia System. Greer, J. (Es.) *Proceedings of 7th World Conference on Artificial Intelligence in Education*, Washington, DC, pp. 351-358.

Perkins, D. N., (1993). Person-Plus: A Distributed View of Thinking and Learning. In Salomon, G. (Ed.) *Distributed Cognitions: Psychological and Educational Considerations*. Cambridge Press, Cambridge University Press, pp. 88-111.

Petre, M. (1995). Why Looking isn't always Seeing: Readership Skills and Graphical Programming. *Communications of the ACM*, **38** (6), pp. 33-44.

Petrie, C.J. (1996). Agent Based Engineering, The Web, and Intelligence. *IEEE Expert*, 11 (**6**), pp. 24- 29.

Piaget, J. (1954). *The Construction of Reality in the Ch*ild. Ballentine Books, New York.

Piaget, J. (1970). *The Science of Education and the Psychology of the Child*. Grossman, New York.

Piaget, J. (1972). *The Principles of Genetic Epistemology*. Basic Books, New York.

Pirolli, P. and Anderson, J. (1985). The Role of Learning from Examples in the Acquisition of Recursive Programming Skills. *Canadian Journal of Psychology*, 39, pp. 240-272.

Pirolli, P. L. and Greeno, J. G. (1988). The Problem Space of Instructional Design. In Psotka, J., Massey, L. D. and Mutter, S. A. (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates Publishers, Hillsdale, NJ., pp. 181-201.

Pirolli, P., and Recker, M. (1994). Learning Strategies and Transfer in the Domain of Programming. *Cognition and Instruction*, **12** (3), pp. 235-275.

Plaza, E., and López de Mántaras, R. (1990). A Case-Based Apprentice that Learns from Fuzzy Examples. Ras, Z., Zemankova, M., Emrich, M.L. (Eds.) *Methodologies for Intelligent System*. 5, pp. 420-427.

Plaza, E. and Arcos J. L., (1993). Reflection and Analogy in Memory-based Learning, *Proceedings of Multistrategy Learning Workshop*, pp. 42-49.

Ploetzner, R., and Fehse, E. (1998). Learning from Explanations: Extending one's own Knowledge during Collaborative Problem Solving by Attempting to Understand Explanations Received from Others. *International Journal of Artificial Intelligence in Education*, **9**(3-4), pp. 193-218.

Podmore, V.N. (1991). 4-Year-Olds, and 6-Year-Olds, and Microcomputers: A Study of Perceptions and Social Behaviours. *Journal of Applied Development Psychology*, **12** (1), pp. 87-101.

Pontecorvo, C. (1993). Developing Literacy Skills Through Co-operative Computer Use: Issues for Learning and Instruction. Duffy, T.M., Lowyck, J., Jonassen, D.H. (Eds.) *Designing Environments for Constructive Learning*. Springer-Verlag, Berlin, pp. 139-160.

Pree, W. 1995. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, Reading, MA.

Preece J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley, Reading, MA.

Pressley, M., Wood, E., Woloshyn, V., Martin, V., King, A., and Menke, D. (1992). Encouraging Mindful use of Prior Knowledge: Attempting to Construct Explanatory Answers Facilitates Learning. *Educational Psychologist*, 27, pp. 91-109.

Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A. (1992). A Multiple-Method Knowledge Acquisition Shell for the Automatic generation of Knowledge-Acquisition Tools. *Knowledge Acquisition*, **4** (2), pp.171-196.

Puppe, F. (1990). *Systematic Introduction to Expert Systems*. Springer-Verlag, Berlin.

Putnam, R. T., Lampert, M., and Peterson, P. L. (1990). Alternative Perspectives on Knowing Mathematics in Elementary Schools. Cazden, C.B. (Ed.), *Review of Research in Education*. American Educational Research Association, Washington, DC., pp. 57-150.

Py, D. (1989). MENTONIEZH: An Intelligent Tutoring System in Geometry. Bierman, D., Breuker, J. and Sandberg, J. (Eds). *Artificial intelligence in Education: Proceedings of the fourth International Conference on Artificial Intelligence in Education*, IOS Press, Amsterdam, pp. 24-26.

Ramadhan, H. (2000). DISCOVER: An Intelligent System for Discovery Programming, *Journal of Cybernetics and Systems*, **31** (1), pp 87-114.

Ramadhan, H. (1992). An Intelligent Discovery Programming System. *Proceedings of ACM Symposium on Applied Computing*, Special Track on Visuality in Computing, pp. 526-532.

Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction*. Elsevier Science Publishing, New York.

Recker, M. M. and Pirolli, P. (1995). Modelling Individual Differences in Students' Learning Strategies. *The Journal of the Learning Sciences*, **4** (1), pp.1-38.

Reigeluth, C. M., & Stein, R. (1983). Elaboration theory. In C. M. Reigeluth (Ed.), *Instructional-design theories and models: An overview of their current status*. Hillsdale NJ: Erlbaum.

Reinhardt, B., and Schewe, S. (1995). A Shell for Intelligent Tutoring Systems. Greer, J. (Ed), *Proceedings of International Conference on Artificial Intelligence in Education*, Association for the Advancement of Computing in Education, Charlottesville, VA, pp. 83-90.

Rickel, J., and Johnson, W. L. (1999). Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence*, 13, pp. 343-382.

Renkl, A., Stark, R., Gruber, H., and Mandl, H. (1998). Learning from worked-out examples: the effects of example variability and elicited self-explanation. *Contemporary Educational Psychology*, 23, pp. 90-108.

Richter, A.M. and Weiss, S. (1991). Similarity, Uncertainty and Case-Based Reasoning in PATDEX. Boyer, R.S. (Eds.), *Automated Reasoning, Essays in Honour of Woody Bledsoe*. Kluwer, pp. 249-265.

Ritter, S. (1997). Communication, Cooperation, and Competition Among Multiple Tutor Agents. In du Boulay, B. and Mizoguchi, R. (Eds.), *Artificial Intelligence in Education*, pp. 31-38.

Ritter, S. and Blessing, S. (1998). Authoring Tools for Component-Based Learning Environments. *Journal of the Learning Sciences*, **7** (1), pp. 107-132.

Roschelle, J., DiGiano, C., Koutlis, M., Repenning, A., Phillips, J., Jackiw, N., and Suthers, D. (1999). Developing Educational Software Components. IEEE Computer Society, Piscataway, NJ., *Computer,* **32** (9), pp. 50-58.

Ritter, S. and Koedinger, K. R. (1997). An Architecture for Plug-in Tutoring Agents. *Journal of Artificial Intelligence in Education*. Association for the Advancement of Computing in Education, Charlottesville, VA, **7** (3/4), pp. 315-347.

Roschelle, J. and Kaput, J. (1996). Educational Software Architecture and Systemic Impact: The Promise of Component Software, *Journal of Educational Computing Researc*h, **14** (3), pp. 217-228.

Roschelle, J., Kaput, J., Stroup, W. and Kahn, T.M. (1998). Scaleable Integration of Educational Software: Exploring The Promise of Component Architectures. *Journal of Interactive Media in Education*. <http://www-jime.open.ac.uk/98/6/>

Rowland, G. (1992). Problem Solving in Instructional Design. Duffy, T.M., and Jonassen, D.H. (Eds.), *Constructivism and the Technology of Instruction*: *A Conversation*. Hillsdale, NJ: Lawrence Erlbaum, pp. 45-55.

Rubtsov, V. (1993). Computer and Learning Activity. Psychological Foundations. *Activity Theory*, 13/14, pp. 4-8.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, NJ.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence*: *A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Ryan, R. (1996). *Self-Explanation and Adaptation*. University of Pittsburgh, Pittsburgh.

Sacerdoti, E. (1977). *A Structure for Plans and Behaviour*. New York, Elsevier North-Holland.

Säljö, R. (1996). Mental and Physical Artifacts in Cognitive Practices. In Reimann, P. and Spada, H. (Eds.). *Learning in Humans and Machines. Towards an Interdisciplinary Learning Science*, London, Pergamon, pp. 83-96.

Salomon, G., and Perkins, D. N. (1998). Individual and Social Aspects of Learning. In Pearson, P. D. and Iran-Nejad, A. (Eds.), *Review of Research in Education*, 23, pp. 1-24.

Schank, R.C. (1994). Tractor Factories and Research in Software Design. *Communications of ACM*, 37, pp.19-21.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press, Cambridge, MA.

Schank, R. C., Fano, A., Bell, B., and Jona, M. (1994). The Design of Goal-Based Scenarios. *The Journal of the Learning Sciences*, **3** (4), pp. 305-345.

Schank, R. C., and Jona, M. Y. (1991). Empowering the Student: New Perspectives on the Design of Teaching Systems. *The Journal of the Learning Sciences,* **1** (1), pp. 7-35.

Schank, R. and Leake, D. (1989). Creativity and Learning in a Case-Based Explainer. *Artificial Intelligence*, **40** (13), pp. 353-385.

Schaefer, B.A., Side, R.S., and Morrison, I.R. (1992) A Knowledge-Based Intelligent Tutoring System: Training Spacecraft Operators, *PC AI Magazine*, 7, 20-25.

Scardamalia, M. and Bereiter, C. (1993). Computer-Support for Knowledge Building Communities. *Journal of the Learning Sciences*, 3, pp. 265-283.

Scardamalia, M., and Bereiter, C. (1996). Adaptation and Understanding: A Case for new Cultures of Schooling. In Vosniadou, S., Corte, E. De, Glaser, R. and Mandl, H. (Eds.), *International Perspectives on the Psychological Foundations of Technology-Based Learning Environments*. Mahwah, NJ: Lawrence Erlbaum. pp. 149-163.

Schoen, D. (1992). Designing as Reflective Conversation with the Materials of a Design Situation. *Knowledge-Based Systems*, **5** (1), pp. 3-14.

Schoen, D. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York, Basic Books.

Schofield, J.W., Eurich-Fulcer, R. and Britt, C.L. (1994). Teachers, Computer Tutors, and Teaching: The Artificially Intelligence Tutor as an Agent for Classroom Change. *American Education Research Journal*, **31** (3), pp. 579-607.

Schwier, R.A. and Misanchuk, E. (1993). *Interactive Multimedia Instruction*. Educational Technology Publications, Cliffs Englewood, NJ.

Sebrechts, M. M., Marsh, R. L. and Furstenburg, C. T. (1990). Integrative Modelling: Changes in Mental Models During Learning. Robertson, S. P., Zachary, W. and Black, J.B. (Eds.) *Cognition, Computing, and Co-operation*. Ablex Publishing, Norwood, NJ, pp. 338-398.

Self, J.A. (1999). The Defining Characteristics of Intelligent Tutoring Systems Research: ITSs Care, Precisely. *International Journal of Artificial Intelligence in Education*, 10, pp. 350-364.

Self, J. A. (1994). Formal Approaches to Student Modelling. Greer, J.E. and McCalla, G.I., (Eds.), *Student Modelling: The Key to Individualised Knowledge Based Instruction*, Springer-Verlag, Berlin, pp. 295-352.

Self, J.A. (1992). Computational Mathematics: The Missing Link in Intelligent Tutoring Systems Research, in Costa, E. (Eds.), *New Directions for Intelligent Tutoring Systems*, Springer-Verlag, Berlin, pp. 295-352.

Self, J. (1990a). Theoretical Foundations for Intelligent Tutoring Systems. *Journal Artificial Intelligence in Education*, **1** (4), pp. 3-14.

Self, J. A., (1990a). Bypassing the Intractable Problem of Student Modelling. Frasson, C. and Gauthier G., (Eds.), *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, Ablex Publishing, Norwood, NJ, pp. 107-123.

Self, J. A. (1989). The case for formalising student models (and intelligent tutoring systems generally). In Bierman, Breuker, D., and Sandberg, J. (Eds.), *Artificial Intelligence and Education: Synthesis and Reflection*, IOS, Springfield, VA, pp. 244.

Self, J. A. (Ed.). (1988). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*. Chapman and Hall, London.

Sein, M. K. and Bostrom, R. P. (1989). Individual Differences and Conceptual Models in Training Novice Users. *Human Computer Interaction*, 4, pp. 197-229.

Sellen A, Nicol A. (1990). Building User-Centred on-line Help. In *The Art of Human-Computer Interface Design*, Laurel, B. (Ed.), Addison-Wesley, Reading, MA, pp.143-153.

Shaw, M. L. and Gaines, B. R. (1989). Comparing Conceptual Structures: Consensus, Conflict Correspondence and Contrast. *Knowledge Acquisition*, **4** (1), pp. 341-364.

Shachter, R., Andersen, S., Szolovits, P. (1994). Global Conditioning for Probabilistic Inference in Belief Networks. *Proceedings of 10th Conference on Uncertainty in Artificial Intelligence*, Seattle WA. Lopez, R., de Mantaras and Poole, D. (Eds.), Morgan Kaufmann Publishers, San Francisco, CA, pp. 514-522.

Shadbolt, N., Motta, E. and Rouge, A. (1993). Constructing Knowledge Based Systems. *IEEE Software*, **10** (6), pp. 34-38.

Shaw, M., and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Upper Saddle River, NJ.

Sheth, A. P., Larson, J. A., Cornelio, A. and Navathe, S. B. (1988). A Tool for Integrating Conceptual Schemata and User Views. *IEEE Data Engineering*, IEEE Computer Society Press, pp. 176-183.

Shiffrin, R.M. and W. Schneider. (1977). Controlled and Automatic Human Information Processing: II. Perceptual Learning, Automatic Attending and a General Theory. *Psychological Review*, 94, pp. 127-190.

Shortliffe, E.H. (1976). *Computer-Based Medical Consultations*: MYCIN. Elsevier Science Publishers, Amsterdam.

Shneiderman, B. (1998). *Designing the User Interface*, Third Edition. Addison-Wesley, Reading, MA.

Shulman, L.S. (1986). Those Who Understand: Knowledge Growth in Teaching. *Education Researcher.* **15** (2), pp.4-14.

Shute, V.J. (1998). DNA - Uncorking the Bottleneck in Knowledge Elicitation and Organisation. Proceedings of Intelligence Tutoring Systemsn Conference, San Antonio, TX, pp. 146-155.

Shute, V. J. (1995). SMART: Student Modelling Approach for Responsive Tutoring. *User Modelling and User-Adapted Interaction*, 5, pp. 1-44.

Shute, V. J. (1994). Learning Processes and Learning Outcomes. In Husen, T. and Postlethwaite, T. N. (Eds.), *International Encyclopedia of Education* (2^{nd}). Pergamon Press, New York, pp. 3315-3325.

Shute, V. J. (1993). A Macroadaptive Approach to Tutoring. *Journal of Artificial Intelligence and Education*, **4** (1), pp. 61-93.

Shute, V. J. and Gawlick-Grendell, L. A. (1994). What does the Computer Contribute to Learning? *International Journal* of *Computers and Education*, **23** (3), pp. 177-186.

Shute, V. J., and Glaser, R. (1990). A Large-Scale Evaluation of an Intelligent Discovery World: Smithtown. *Interactive Learning Environments,* **1** (1), pp. 51-77.

Shute, V. J. and Psotka, J., (1996). Intelligent Tutoring Systems: Past, Present, and Future. In Jonassen, D. (Ed.), *Handbook of Research for Educational Communications and Technology*, Macmillan, New York, NY., pp. 570-600.

Shute, V. J., and Regian, W. (1993). Principles for Evaluating Intelligent Tutoring Systems. *Journal of Artificial Intelligence in Education,* **4** (2/3), pp. 245-272.

Skalak, C.B, and Rissland, E. (1992). Arguments and Cases: An Inevitable Twining. *International Journal of Artificial Intelligence and Law,* **1** (1), pp.3-48.

Skinner, B.F. (1953). *Science and Human Behaviour.* New York, Macmillan.

Siemer, J., and Angelides, M.C., (1998). Towards an Intelligent Tutoring Architecture that Supports Remedial Tutoring. *Artificial Intelligence Review* 12, pp. 469-511.

Siemer, J., S., Taylor, J.E. and Elliman, T. (1995). Intelligent Tutoring Systems for Simulation Modelling in the Manufacturing Industry. *International Journal of Manufacturing Systems Design*, **2** (3), pp.165-175.

Silverman, B.G. (1992). *Critiquing Human Error - A Knowledge Based Human-Computer Collaboration Approach.* Academic Press, London.

Sim, I., and G. Rennels (1995). Developing A Clinical Trial Ontology: Comments on Domain Modelling and Ontological Reuse, Knowledge Systems Laboratory Medical. *Computer Science*, pp.95-60.

Sime, J. A. and Leitch, R.R. (1993). A Specification Methodology for Intelligent Training Systems, *Computers in Education*, 20, pp. 73-80.

Sleeman, D., Hirsh, H., Ellery, I., and Kim, I. (1990). Extending Domain Theories: Two Case Studies In student Modelling. *Machine Learning*, 5, pp. 11-37.

Sleeman, D. and Brown, J. S. (1982). *Intelligent Tutoring Systems*. Academic Press, New York.

Smith, J. P., diSessa, A. A., and Roschelle, J. (1993). Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences*, **2** (2), pp. 15-164.

Soloway, E. and Ehrlich, K. (1984). Empirical studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, **10** (5), pp.595-609.

Soloway, E., Guzdial, M., Brade, K., Hohmann, L., Tabak, I., Weingrad, P. and Blumenfeld, P. (1991). Technical Support for the Learning and Doing of Design. Jones, M. and Winne, P.H. (Eds), *Adaptive Learning Environments*: *Foundations and Frontiers*, Springer-Verlag, Berlin, pp. 173-200.

Sommerville, I. (1996). *Software Engineering*. Fourth Edition, Addison-Wesley. Reading, MA.

Sparks, R. Dooley, S., Meiskey, L. and Blumenthal, R. (1999). The LEAP Authoring Tool: Supporting Complex Courseware Authoring Through Reuse, Rapid Prototyping, and Interactive Visualizations. *International Journal of Artificial Intelligence in Education*, **10** (1), pp. 75-97.

Specht, M., Weber, G., Heitmeyer, S. and Schöch, V. (1997). AST: Adaptive WWW-Courseware for Statistics. In Brusilovsky, P., Fink. J., and Kay, J. (Eds.) *Proceedings of 6th International Conference on Adaptive Systems and User Modeling on the World Wide Web*, Chia Laguna, Sardinia, Italy, pp. 91-95.

Stead, W. W., Miller, R. A.,. Musen, M. A and Hersh. W. R. (2000). Integration and Beyond: Linking Information from Disparate Sources and into Workflow. *Journal of the American Medical Informatics Association*, **7** (2), pp. 135-145.

Spewak, S. (1992). *Enterprise Architecture Planning*. John Wiley, New York.

Steels, L. (1993). The Componential Framework and its Role in Reusability. In David, J. M., Krivine, J.P., and Simmons, R. (Eds.), *Second Generation Expert Systems*, Spinger-Verlag, Berlin, pp. 273-298.

Steels, L. (1992). Reusability and Knowledge Sharing. In Steels, L. and Lepape, B. (Eds.), Enhancing the Knowledge Engineering Process: Contributions from ESPRIT, Elsevier, Amsterdam, pp. 240-271.

Steels, L. (1990). Components of Expertise, *AI Magazine*. 11, pp. 30-49.

Stepanov, A. and Lee, M. (1995). The Standard Template Library.

http://www.cs.rpi.edu/~musser/doc.ps.

Stern, M., Woolf, B. P., and Kuroso, J. (1997). Intelligence on the Web? In Boulay, B. d. and Mizoguchi, R. (Eds.) *Proceedings of 8th World on Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, Kobe, Japan, Amsterdam, IOS, pp. 490-497.

Stuart K. Card, Jock Mackinlay, and Ben Shneiderman. (1998). Readings in Information Visualisation - *Using Vision to Think.* Morgan Kaufmann, Palo Alto, CA.

Strube, G. (1991). The Role of Cognitive Science in Knowledge Engineering. Schmalhofer, F., and Strube, G. (Eds.), Proceedings of *First joint Workshop on Contemporary Knowledge Engineering and Cognition.* Springer-Verlag, Berlin, pp. 161-174.

Studer, R., Benjamins, V.R., and Fensel, D. (1998). *Knowledge Engineering,Principles and Methods.* In *Data and Knowledge Engineering* **25** (1-2), pp. 161-197.

Studer, R., Fensel, D., Decker, S. and Benjamins, V.R. (1999). Knowledge Engineering: Survey and Future Directions. In Puppe, F. (Ed.), Knowledge-Based Systems: Survey and Future Directions, *Proceedings of the Fifth Conference on Knowledge-based Systems, Lecture Notes in Artificial Intelligence*, Wuerzburg, Springer-Verlag, Berlin, pp. 1-23.

Sun Microsystems. (1999). *Java Look and Feel Guidelines.* Addison-Wesley, Reading, MA.

Suthers, D. and Jones, D. (1997). An Architecture for Intelligent Collaborative Educational Systems. In Boulay, B. d. and Mizoguchi, R. (Eds.), *Proceedings of Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, Kobe, Japan, Amsterdam, IOS, pp. 55-62.

Sweller, J. (1989). Cognitive technology: Some Procedures for Facilitating Learning and Problem Solving in Mathematics and Science. *Journal of Educational Psychology*, **81** (4), 457-466.

Sycara, K. (1988). Using Case-based Reasoning for Plan Adaptation and Repair. In Kolodner, J. (Ed.), *Proceedings Case-Based Reasoning Workshop, DARPA*, Morgan Kaufmann, Pal Alto, pp. 425-434.

Szyperski, C. (1997). *Component Software - Beyond Object-Oriented Software.* Adison-Wesley, Reading, MA.

Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7, pp. 83-124.

Taylor, R. N., Medvidovic, N., Anderson, K., Whitehead, Jr., E. J., Robbins, J. E., Nies, K. A., Oreizy, P., and Dubrow, D. L. (1996). A Component and Message Based Architectural Style for GUI Software. *IEEE Transaction Software Engineering*, **22** (6), pp.390-406.

Tessmer, M. (1993). *Planning and Conducting Formative Evaluations.* London, Kogan Page.

Thorndike, E. (1932). *The Fundamentals of Learning.* Teachers College Press.

Tobin, K. G. (1993). *The Practice of Constructivism in Science Education.* American Associate for the Advancement of Science Washington, D. C.

Top J., and Akkermans, H. (1994). Tasks and Ontologies in Engineering Modelling, *International Journal of Human-Computer Studies*, 41, pp. 585 - 617.

Towne, D.M. (1997). Approximate Reasoning Techniques for Intelligent Diagnostic Instruction. *International Journal of Artificial Intelligence in Education*. **8**, (3-4), pp. 262-283.

Tu, S. W., Eriksson, H., Gennari, J. H., Shahar, Y., and Musen, M. A. (1995). Ontology-Based Configuration of Problem Solving Methods and Generation of Knowledge; Acquisition Tools: Applications of PROTÉGÉ-II to Protocol Based Decision Support. *Artificial Intelligence in Medicine*, 7, pp. 257-289.

Tulving, E. (1977). Episodic and Semantic Memory. Tulving, E., and Donaldson, W. (Eds.) *Organisation of Memory*. Academic Press, Boston, MA, pp. 381-403.

Twidale, M., Pengelly, M., Chanier, T., and Self, J. (1992). Experiments on Knowledge Acquisition for Learner Modelling. Cerri, S.A. and Whiting, J. (Eds.) *Learning Technology in the European Communities,* Kluwer, Dordrecht, pp. 355-368.

van Heijst, G., Schreiber, A.T., and Wielinga, B.J. (1997). Using Explicit Ontologies in Knowledge Based Development, *International Journal of Human and Computer Studies*, 46, pp. 293-310.

VanLehn, K. (1999). Rule-Learning Events in the Acquisition of a Complex Skill: An Evaluation of Cascade. *The Journal of the Learning Sciences*, **8** (1), pp. 71-125.

VanLehn, K. (1996a). Conceptual and Meta Learning During Coached Problem Solving. In Frasson, C., Gauthier, G., and Lesgold, A. (Eds.), *Proceedings of the Third International Conference on Intelligent Tutoring Systems,* Springer-Verlag, New York. pp. 29-47.

VanLehn, K. (1996b). Cognitive Skill Acquisition. In Spence, J., Darly, J., and Foss, D. J. (Eds.), *Annual Review of Psychology*, Palo Alto, CA., 47, pp. 513-539.

VanLehn, K., and Jones, R. M. (1993). Learning by Explaining Examples to Oneself: A Computational Model. In Chipman, S. and Meyrowitz, A. (Eds.), *Cognitive Models of Complex Learning* Kluwer Academic Publishers, Boston, MA, pp. 25-82.

VanLehn, K., Jones, R. M., and Chi, M. T. H. (1992). A Model of the Self Explanation Effect. *The Journal of the Learning Sciences*, **2** (1), pp. 1-59.

VanLehn, K. (1991). Rule Acquisition Events in the Discovery of Problem-Solving Strategies. *Cognitive Science*, 15, pp. 1-47.

VanLehn, K. (1988). Student Modelling. In Polson, M. (Ed.), *Foundations of Intelligent Tutoring Systems*. Lawrence Erlbaum, Hillsdale, NJ, pp. 55-78.

Van Marcke, K. (1998). GTE: An Epistemological Approach to Instructional Modelling. *Instructional Science*, 26, pp. 147-191.

Van Marcke, K. (1992). Instructional Expertise. In Frasson, C., Gauthier, G., and McCalla, G.I. (Eds.) *Proceedings of Intelligent Tutoring Systems*. Springer-Verlag, New York, pp. 234-243.

Van Merrienboer, J. and Krammer, H. (1992). A Descriptive Model of Instructional Progresses in Interactive learning Environments for Elementary Computer Programming. In Dijkstra, S., Krammer, H. and Van Merrienboer, J. (Eds.), *Instructional Models in Computer-Based Learning Environments*, Spring-Verlag, Berlin, pp. 213-228.

Vassileva, J., (1997). A New View of Interactive Human-Computer Environments. In Anthony Jameson, Cecile Paris, and Carlo Tasso (Eds.) User Modelling, *Proceedings of the Sixth International Conference, on* User *Modelling*. Vienna, New York, Springer Wien New York, pp. 433-435.

Villano, M. (1992). Probabilistic Student Models: Bayesian Belief Networks and Knowledge Space Theory. In *Proceedings of the Second International Conference on Intelligent Tutoring Systems*, Springer-Verlag, Berlin, pp. 491-498.

Visser, W. (1990). More or Less Following a Plan During Design: Opportunistic Deviations in Specification. *International Journal of Man-Machine Studies*, pp. 247-278.

Vonk, K. (1993). Mentoring the Beginning Teacher. *Mentoring*, **1** (1), pp. 31-41.

Voss, J., Wiley, J., and Carretero, M. (1995). Acquiring Intellectual Skills. *Annual Review of Psychology*, 46, pp. 155-181.

Vygotsky, L. (1978). *Mind in Society*: *The Development of Higher Psychological Processes.* Harvard University Press, Cambridge, MA.

Wadsworth, B.J. (1971). *Piaget's Theory of Cognitive Development*. Longman, New York.

Walther, E., Eriksson, H., and Musen, M.A. (1992). Plug and Play: Construction of Task Specific Expert System Shells Using Sharable Context Ontologies. *Proceedings of AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition*, San Jose, CA, pp.191-198.

Wasson, B. (1996). Instructional Planning and Contemporary Theories of Learning: Is this a Self-Contradiction? In Brna, P., Paiva A. and Self, J. (Eds.) Proceedings of the_*European Conference on Artificial Intelligence in Education,* Lisbon: Colibri, pp. 23-30.

Wasson, B. (1992). PEPE: A Computational Framework for a Content Planner. In Dijkstra, S.A., Krammer, H.P.M and van Merrienboer, J.J.G. (Eds.), *Instructional Models in Computer-Based Learning Environments*. Sringer-Verlag, New York, 104, pp. 153-170.

Waterworth, J.A. (1992). *Multimedia Interaction with Computers: Human Factors Issues.* Ellis Horwood, Chichester, Sussex.

Webb, N. (1982). Student Interaction and Learning in Small Group. *Review of Educational Research*, 52, pp. 421- 445.

Weber, G and Sprecht, M. (1997). User Modelling and Adaptive Navigation Support in WWW - Based Tutoring Systems. In Jameson, A., Paris, C. and Tasso, C. (Eds.), In Proceedings of the *Sixth International Conference on User Modelling* User Modelling, Vienna, Sprinter Wien, New York, pp. 289-300.

Wedman, J., and Tessmer, M. (1993). Instructional Designers' Decisions and Priorities: A Survey of Design Practice. *Performance Improvement Quarterly*, **6** (2), 43-57.

Weld, D. S. (1994). An Introduction to Least Commitment Planning. *AI Magazine*, **15** (4), pp.27-61.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann, Los Altos, CA.

White, B. Y. and Frederiksen, J. (1985). QUEST: Qualitative Understanding of Electrical Troubleshooting, *ACM SIGART Newsletter*, 93, pp. 34-37.

Wielinga, B. J. and Breuker, J. A. (1990). Models of Expertise. *International Journal of Intelligent Systems*, 5, pp. 497 - 509.

Wielinga, B., Sandberg, J. and Schreiber, G. (1997). Methods and Techniques for Knowledge Management: What has Knowledge Engineering to offer, *Expert Systems with Applications*, **13** (1), pp. 73-84.

Wielinga, B. J., Schreiber, A. Th., and Breuker, A. (1992). KADS: A Modelling Approach to Knowledge Acquisition. *Knowledge Acquisition*, **4** (1), pp. 5-54.

Wielinga, B.J., Van de Velde, W., Schreiber, A.T., and Akkermans, J.M. (1993). Towards a Unification of Knowledge Modelling Approaches. David, J.M., Krivine, J.P. and Simmons, R. (Eds.). *Second Generation Expert Systems*, Springer-Verlag, Berlin, pp. 299-335.

Willis, J. (1995). A Recursive, Reflective Instructional Design Model Based on Constructivist-Interpretivist Theory. *Educational Technology*, **35** (6), pp. 5-23.

Wilson, B., and Cole, P. (1992). A Critical Review of Elaboration Theory. *Educational Technology Research and Development*, **40** (3), pp. 63-79.

Winkels, R., J. Sandberg, and J. Breuker. (1990). The Coach. *Developing Intelligent Help Systems*, Breuker, J. (Ed.), Copenhagen, EC, pp. 119-146.

Woolf, B. (1992). Building knowledge Based Tutors. In Tomek, I. (Ed.), *Proceedings of the Fourth International Conference on Computer Assisted Learning*, Wolfville, NS, pp. 46-60.

Woolf, B. P. (1988). Intelligent Tutoring Systems: A Survey. In Schrobe, H. (Ed.), *Exploring Artificial Intelligence,* Morgan Kaufmann, Palo Alto, CA, pp. 1-44.

Woolf, B. and Cunningham, P. A. (1987). Multiple Knowledge Sources in Intelligent Tutoring Systems. *IEEE Expert*, 2 (2), pp. 41-54.

Woolf, B., and Hall, W. (1995). Multimedia Pedagogues - Interactive Systems for Teaching and Learning. *IEEE Computers*, pp.74-80.

Woolf, B. and McDonald, D. D. (1984). Building a Computer Tutor: Design Issues, *Computer*, **17** (9), pp. 60-73.

Wong, W.K. and Chan, T.W. (1997). A Multimedia Authoring System for Crafting Topic Hierarchy, Learning Strategies, and Intelligent Models. *International Journal of Artificial Intelligence in Education,* **8**, (1), pp. 71-96.

World Wide Web Consortium (W3C), (1998). *Extensible Markup Language (XML) 1.1.* Available from http://www.w3.org.

World Wide Web Consortium (W3C), (1998). *Precision Graphics Mark-up Language* (PGML). Available from http://www.w3.org.

World Wide Web Consortium (W3C) (1999). Scalable Vector Graphics (SVG) 1.0 *Specification*: Available from http://www.w3.org.

Yang, Q. (1997). Intelligent Planning: A Decomposition and Abstraction Based Approach. *Artificial Intelligent.* Springer Verlag, Berlin, pp. 163-188.

Yazdani, M. (1987). *Intelligent Tutoring Systems: An Overview, In Artificial Intelligence and Education.* Laeler, R. and Yazdani, M. (Eds.), Ablex Publishing, Norwood, pp. 182-201.

Yazdani, M. and Lawler, R. W. (1986). Artificial Intelligence and Education: An Overview. *Instructional Science*, 14, pp. 197-206.

Young, R. M. (1983). Surrogates and Mappings: Two Kinds of Conceptual Models for Interactive Devices. Genter, D., and Stevens, A. (Eds.) *Mental Models.* Lawrence Erlbaum, Hillsdale, NJ., pp. 35-52.

Zeigler, B. P. (1991). Object-Oriented Modelling and Discrete-Event Simulation. *Advances in Computers*, 33, pp. 67-114.

Zsambok, C.E., and Klein, G. A. (1997). *Naturalistic Decision Making.* Lawrence Erlbaum, Mahwah, NJ.

APPENDICES

APPENDIX A

BACKGROUND RESEARCH MATERIALS

A1.0 Instructional Theories

The use of "intelligent machines" for education was started by Pressey (1926) to develop an instructional machine with multiple-choice questions and answers. The system delivered questions and then provided immediate feedback to each learner. The problem with this "machine" is that it was not adaptive to the needs of the user because of the ways that knowledge was represented.

Educational psychology began to develop learning processes and models in the 1950s. The pioneering work postulates a cognitive development approach (Bloom, 1956), (Carroll, 1963). Cognitive development takes place by either assimilating a new learning relationship directly into the present schema by extending it to subsume the new relationship or by creating a new schema (Piaget, 1971). This was followed by those that conceptualised learning as the process of reinforcement of stimulus–response connection (Skinner, 1957), (Glaser, 1976), i.e., changes in behaviour are the result of learners' response to events (stimuli) that occur in the environment. These pioneering programmes were not flexible and they were not adaptable to any domain.

"Authoring languages" development started in the early 1970s to allow non-programmers to develop computer-assisted learning programmes. Use of artificial intelligence (AI) techniques subsequently followed this period. Artificial intelligence techniques involve the use of models of human cognition and intelligent tools. ITS are developed by combining both components within a domain. Research in ITS began as an attempt to provide solutions to the shortcomings of the computer-based learning programmes. Developing an ITS is usually knowledge intensive and requires that the developer must to be an expert in that domain discipline (e.g., simulation modelling or biology).

Computer-based training (CBT) and computer-aided instruction (CAI) were the first such systems deployed as an attempt to teach using computers. Although both CBT and CAI may be somewhat effective for educational use, they do not provide the same kind of individualised attention that a

student would receive from a human tutor (Bloom, 1984). For a computer-based educational system to provide such attention, it must reason with the domain and the learner. This has prompted research in the field of ITS. ITS offer considerable flexibility in presentation of material and a greater ability to respond to idiosyncratic student needs.

A1.1 Theory of Instruction

Theories of instruction can be broadly classified into three categories:

i. *Behavioural and Associationist theories* of learning focus on drives, responses, stimuli and rewards. Drives and stimuli impel animals to generate responses and rewards, and causes certain stimulus–response connections to be strengthened. Despite the seemingly low-level nature of behavioural theories of learning, some quite sophisticated high-level theories of learning are based on behavioural foundations. Landmark works include those by Anderson (1987), Skinner (1957) and Gagne (1956).

ii. *Cognitive theories* of learning focus on knowledge representation and epistemology, as well as human information processing and problem-solving strategies. Of primary importance is understanding the knowledge structures and processes that underlie competent performance on particular tasks in some domain, the initial knowledge state of a learner, and the processes by which a learner is transformed from novice to expert (Eliot and Woolf, 1995). Predicted misconceptions and errors are used to evaluate the plausibility of competing theories.

iii. *Meta-cognitive theories* of learning focus on the cognitive processes that allow learners to self-monitor and self-regulate their thinking. This group, in particular, emphasises the important role that social (social cognition) and environmental interactions (situated cognition) play in fostering the development of thinking and learning skills. Constructivism (McLellan, 1996), active learning, apprenticeship, communities of practice, and portfolio-based assessment are key concepts in the full range of meta-cognitive theories of learning.

The most dominant and long-lasting approach for the delivery of an instructional system is to conceptualise learning as the process of reinforcement of stimulus–response associations. Thorndike's (1882), study of animal intelligence is a landmark in the literature of psychology and learning. He proposed that training was a result of gradual strengthening of associations between a stimulus (S) and a response (R). Thorndike proposed the law of effect, which states that "*responses having favourable consequences will be learned*".

Brown (1990), Anderson (1992), Corbett and Anderson (1990) have explained how current instructional theories and architectures in ITS have impacted learning theories, and their environment. Brown (1990) makes the distinction between explicit cognition (practice) and implicit cognition (theories), and differentiates between the ways that students learn in school (by using formal reasoning and abstraction) and outside school (by experience, demonstration, problem solving in practical situations). The analogy is extended further to everyday cognition (typically those types of processes in which one would engage in the course of ordinary living) and expert cognition (the ability to abstract problems in order to solve them). These theories are currently being used as the basis for the design of ITS. Cognitive theories involve the behaviour of the student during instruction, their problem-solving methods, support shared conversations and investigations; allow issues and problems to emerge from the investigative activities and allow the reflective process of human mental modelling in problem solving (Brown, 1990).

Brown's assertion that theories of learning and ITS design are related is supported by Anderson (1990), who believes that there is considerable research in cognitive psychology that can be used as a guide to the development of ITS. He states that the success of an ITS can depend on its ability to achieve "task decomposition", a process of simplifying learning, and the monitoring of a student's belief by generating production rules (task analysis) for a restricted knowledge domain.

Piaget (1970) is best known for his work on elucidating the early stages of childhood development. Discrete stages of competency fits well with the cognitive learning theories notions of novice/expert differences in representation and reasoning capabilities. His view of a child as a scientist, trying to make sense of the world and actively constructing knowledge, as opposed to simply memorising it, is an extreme departure from typical behavioural theories.

Instructional theories are essential in the design of ITS. These theories help to understand the learning processes and help in the design of effective instruction. One of the most important features of a CAI program is its interactivity, and the computer is used as the vehicle through which certain teaching techniques may be used and enhanced. ITS programs are knowledge centred as they are based on a well-defined model, and ITS developers are interested in a set of instructional principles general enough to apply to a variety of teaching domains. Thus, ITSs have developed from instructional theories, and research in this field helps understand the relationship between the learner, the content and the instructional strategies.

The *ACT* (*Adaptive Control of Thought*) and the ACT-R theory of learning and problem solving is concerned primarily with the acquisition of cognitive skills (Anderson, 1983), (Anderson, 1993). This theory provides a distinction between procedural, and declarative knowledge. Declarative knowledge is knowledge that is factual in nature, and can be made explicit, whereas procedural knowledge is knowledge about how to accomplish some task, such as driving a car or troubleshooting a circuit (Patrick, 1992). Basically, Anderson's theory states that skill acquisition occurs when declarative knowledge is converted, or compiled to procedural knowledge (productions) through practice, and gradually proceeds to procedural knowledge before it is internalised. Declarative knowledge about the task may take the form of basic facts about the domain, simple procedures, or specific procedures. Procedural knowledge is acquired by making inferences to already acquired domain facts or procedures and practice (Anderson, 1987).

The second step of the skill acquisition process, is *knowledge compilation*, which involves compilation and proceduralisation. Compilation is the classification of a sequence of productions into a single production that has the same effect as the sequence (chunking). Proceduralisation is the instantiation of variables in a production, to essentially create a more specialised production, thereby eliminating retrieval from long-term memory retrieval (Anderson,1988). The proceduralisation process is analogous to Shiffrin's automatisation process (Shiffrin, 1977), whereby the performance of skills is learned in an automatic fashion, eliminating the cognitive load required for performing the skill. Once the declarative knowledge has been converted to procedural knowledge, it is further refined in the third stage of skill acquisition: tuning. The tuning stage of skill acquisition consists of three phases: generalisation, discrimination, and strengthening.

Generalisation is essentially the process of replacing bound facts in a production with variables to broaden the production's scope of applicability. For example, this can have the effect of eliminating productions when two or more productions have identical consequence and the generalisation process results in the productions having matching antecedents. Discrimination is the addition of antecedents to a production, which has the effect of narrowing the scope of the production. Finally, strengthening is the process whereby competing productions are weighted based upon feedback as to their applicability (reliability). Anderson theorises that positive feedback is a more gradual process than negative feedback; in other words, a production gets slowly promoted over time as it proves correct, whereas a production will be quickly demoted if it proves incorrect. Anderson's theory has been successfully applied to other ITS (Fink, 1990), (Wenger, 1990), (Anderson et al., 1987).

It can be inferred that many instructional theories that can be considered too simplistic to apply to the broad scope of "education" can be adapted to the more limited scope of "Instruction" quite well. This section describes some of the theories used and adapted for this research.

Gagne (1988) postulates that learning tasks for intellectual skills can be organised in a hierarchy according to complexity: stimulus recognition, response generation, procedure following, use of terminology, discriminations, concept formation, rule application and problem solving. The primary significance of the hierarchy is to identify prerequisites that should be completed to facilitate learning at each level. Prerequisites are identified by doing a task analysis of a learning/training task or by transforming a pedagogical hierarchy into a task classification structure. The task classification structure provides a basis for the sequencing of instruction. According to Gagne, the best method for facilitating learning in this range is practice with feedback, and suggestive information on how to make a proper response (Patrick, 1992). A second aspect of Gagne's theory are his nine instructional events (Gagne, 1985), and corresponding cognitive processes, which can serve as a general purpose pedagogical structure for presenting information to the student: (i) Gaining attention (reception), (ii) Informing learners of the objective (expectancy), (iii) Stimulating the recall of prior learning (retrieval), (iv) Presenting the stimulus (selective perception), (v) Providing "learning guidance" (semantic encoding), (vi) Eliciting performance (responding), (vii) Providing feedback (reinforcement), (viii) Assessing performance (retrieval), and (ix) Enhancing retention and transfer (generalisation).

These events should satisfy or provide the necessary conditions for learning and serve as the basis for designing instruction and selecting appropriate media (Gagne et al., 1992). To design instruction using the Gagne and Briggs model of instructional design requires that learning outcome are categorised and instructional events organised for each kind of learning outcome.

Merrill's Component Display Theory (CDT), (Merrill, 1983), (Merrill, 1996) is based on the same assumptions as Gagne's theory that different classes of learning outcomes require different procedures for teaching and assessment. CDT is concerned with teaching individual concepts or principles, classifies objectives on two dimensions and formats instruction to provide learner control.

Using this framework of knowledge, Merrill postulates four different types of training material presentation strategies: (i) Expository general (telling a rule), (ii) Expository instance (telling an

example), (iii) Inquisitor general (asking about a rule), and (iv) Inquisitor instance (asking about an example).

Thus, a complete lesson would consist of an objective followed by some combination of rules, examples, recall, practice, feedback, helps and mnemonics appropriate to the subject matter and learning task. Indeed, the theory suggests that for a given objective and learner, there is a unique combination of presentation forms that results in the most effective learning experience. Both the knowledge framework and presentation strategies are used in the instructional design strategies for this research. The specific details of how these theories are applied are discussed in Section 2.3.

A1.2 Cognitive Bases of Instruction

Instruction is generally affected by theoretical models in the philosophy of science describing how scientific research advances scientific knowledge and models in psychology and epistemology describing human learning processes. For example, Kuhn's (1966) philosophy of science and Piaget's theory (1970) of cognitive development have been drawn by various researchers and used by some as a basis for curricular development. Because these fields both deal with human knowledge acquisition, their related ideas and similarities in their taxonomies, are often used for the purposes of designing "a philosophically more valid science curriculum" (Hodson, 1988), or a "generative learning model" (Osborne and Wittrock, 1985).

However, in its attempt to offer a solution to instructional design problems, the systems approach generated a set of its own problems and accompanying critics. Ironically, models based on information processing theory and systems theory are now called "traditional" design models. Jonassen (1991) criticises systematic models as a "top-down" behaviourist and subject-matter-expert approach to education. Instead, he champions constructivist instructional approaches (Jonassen, 1994). Wedman and Tessmer (1993) comment that systematic models are too linear and time-consuming to be practical in the "real world" and Rowland (1992) states that systematic models do not reflect the ways that instructional design experts really work when designing learning materials.

From the learner's point of view, the weakness of the systems approach for instructional design becomes apparent: learning may not always be linear. It may require the learner to spontaneously adopt a different approach for learning the task at hand. Some critics complain that linear

instruction is often boring and lacks creativity although proponents try to refute that charge (Dick, 1995).

From the developer's point of view, instructional design cannot always be based on careful and logical decomposition of the knowledge and skills to be learned (Carroll, 1992). "One size fits all" does not work successfully for instructional purposes. Development does not occur in a linear fashion; some steps require several visits and some may follow a different order or be completely eliminated. Too many instructional objectives with too much detail may be counterproductive, especially if they are created early in the development process (Willis, 1995).

Piaget's theory of genetic epistemology (Wadsworth, 1971), (Piaget, 1972) describes the constraints upon the development of knowledge in terms of cognition and cognitive development. This theory played a large role in the critique of Skinner's (Skinner, 1968) behaviourism in favour of cognition as the operating paradigm for instructional development.

Piaget's model describes the mind as organising internalised regularities (operations) into dynamic cognitive structures known as schema, which represent the relationships between perceived environmental regularities (concepts). According to Piaget, cognitive development can proceed in one of two ways; either by assimilating a new relationship directly into the present schema by extending it to subsume the new relationship "simple growth" (Wadsworth, 1971), or by creating a new schema (or changing the structure of the original schema to create a different schema) by accommodation. Assimilation is said to reflect a quantitative change in mental structure (growth) whereas accommodation reflects a qualitative one (development). The balance between accommodation and assimilation is known as "equilibrium". Assimilation and accommodation are important factors for instructional design and account for all cognitive growth and development, and therefore shape human learning.

Vygotsky (1978) introduced the notion of a zone of proximal development in which a learner could interact effectively with a facilitator to acquire new competencies. The zone of proximal development refers to the zone between the things one can already do and the things it would be foolhardy to attempt. The middle range are those things that one can reasonably hope to master either on our own, with the right tools or cognitive artefacts, or with the right facilitators. Much of the work in situated cognition has its roots in this work.

Despite the large number of innovative ideas expressed in the classic works of these scholars, behavioural theories of learning to date have had the largest impact on the practice of education

and instruction. The current development paradigm is in the direction of the cognitive and meta-cognitive camps.

Skinner (1953) applied his ideas of "*operand conditioning*" to classroom teaching. This approach has quickly been adapted for the technique of programmed text to employee training, including some sophisticated methods of computer-assisted instructions (Orione and Rummler, 1987). Because recognising positive reinforcement is central to learning, an instructional system must incorporate these principles at an early stage in courseware development.

There is a growing emphasis on the cognitive load of the learner and the mental processes, including attention, memory, language, reasoning and problem solving. As Howell and Cooke (1989) observed, the changes in technology have increased the demands on the learner who now, instead of performing simple procedural and predictable tasks, must become responsible for inferences, diagnosis, judgements and decision making often under severe time pressure.

Bruner (1961) defines one of the most coherent and consistent cognitive descriptions of learning. It sees learning not merely as a passive unit elicited by stimulus and strengthened or weakened by reinforcement, but as an active process in which the learner infers principles and rules and then tests them out. Learning is something they themselves make happen by the manner in which they handle incoming information and put them into use. Bruner helped popularise the notion of alternative modes of representation (actions, pictures, and symbols) tied to developmental stages. Representing knowledge in all three modes supports deeper understanding of material, such that problem solving could be done efficiently in the most appropriate mode and difficulties occurring in one mode could be overcome by falling back to a more fundamental mode. Bruner (1961, 1990) has expanded his theoretical framework to encompass the social and cultural aspects of instruction.

Bruner and Anglian (1973), suggested that three important variables are important for learning to take place – the learner, the knowledge to be learned, and the learning process. These variables provide ways of arranging tasks in relation to practical aspects of learning. For learning to take place, the tasks involved must be associated and structured so that the student can easily grasp them.

Keeler (1974) produced a student-centred teaching program, which was based on behavioural principles, and suggested that the material be taught in units. It also suggested that tutorial help should be given during instruction, and that the learner takes a diagnostic test at the end of the unit. This epistemology should be applied during courseware design stages.

Livingstone and Borko (1989) suggested that an experienced practitioner has a more sophisticated understanding of practice than a novice. Mentees have cognitive schemas that are less elaborate, less accessible and less interconnected when brought to bear on professional practice. Similarly, recent studies have shown that, for the purpose of teaching, novices have to relearn their subject matter (Vonk, 1993). As Borko et al. (1988) note, all teachers need to reshape and revise their knowledge of subject content. Shulman (1986), suggested that teachers have to develop a "pedagogical content knowledge" in order to translate academic knowledge into school knowledge.

Piaget (1970) identified four factors that are necessary for a theory of cognitive development. These factors include maturation, experience with physical environment, social experience and equilibrium or self-regulation. What is of interest to this research is the importance he places on peer interaction. He suggested that peer interaction induces cognitive conflict, which in turn results in cognitive restructuring and development. Piaget believed that the students learn more working in a group than they would from interaction with a lecturer.

Vygotsky's (1979) approach also rejected the strict separation of the individual and his/her social environment. He treats cognitive development as a process of acquiring culture. He suggests that social interaction plays a fundamental role in the development of cognition. He points out how peer interaction enhances the development of logical reasoning through a process of active cognitive reorganisation. It helps the individual to acknowledge and integrate a variety of perspectives on a problem and this process of co-ordination, in turn, produces superior intellectual results. He argues that it is not really how the learner takes overall responsibility that had formerly been vested in the adult, but how the learner begins to develop a definition of the task situation that will allow him/her to participate in the communicative context. He sees the development of the task of reasoning as taking a top-down course.

Vygotsky gives much insight into assessing the “developmental level” of a learner. Because learning can result in the change of the behaviour of the learner, he characterised behavioural change in terms of shifts in control or responsibility. He described shifting control within activities as a "zone of proximal development (ZPD)". He said the ZPD was the difference between "actual development as determined by problem solving" and the higher level of "potential development as determined through problem solving under guidance or in collaboration". This concept is of great importance to the development of instructional systems as it involves the assessment of the learner's cognitive ability and evaluation of instructional practices.

One important aspect of the cognitive emphasis is that it focuses on the behaviour to be learned and suggests that different approaches might be used to support learning for different behaviours. One good illustration of what is to be learned is presented by Gagne's (1984) system. In this model, when teaching an intellectual skill, the trainer is supposed to present an example of the concept rule: when teaching problem solving, the trainer presents novel problems. Gagne's theory begins with a framework of learning outcomes considered essential for an understanding of human learning as it occurs in instructional settings. The learning outcomes are treated as acquired capabilities and are grouped into five categories: verbal information, intellectual skills cognitive strategies, motor skills and attitudes.

The theory proposes that each of the categories of learning outcome requires a different set of conditions for optimising learning, retention and transferability. Optimal conditions are defined both in terms of the instructional environment and the learner's memory. The process of learning involves using attention, selective perception, short-term memory, rehearsal, long-term memory, storage and retrieval of previously learned information, and reinforcement by external feedback.

Ausubel (1978) applied a cognitive approach to learning based on assimilation theory. The theory suggested that meaningful learning results from the interaction between new concepts, which learners acquired, and the cognitive structures he possesses. Instructional systems designer must consider what knowledge the learner possesses and structure its curriculum to facilitate learning.

In a review of training, Goldstein (1980) defined training as, "the acquisition of skills, concepts or attitude that results in improved performance in an on-the-job situation". Instructional systems are therefore intimately concerned with the theories or principles of learning and cognitive skill acquisition (Greeno et al., 1998), (VanLehn, 1996b), (Glaser and Bassock, 1989), (Voss, et al., 1995). Bramley (1990) defined training as, "systematic development in individual to perform task". This implies that training should be planned and controlled, and task performance is the criterion of success.

A2.0 Intelligent Tutoring Systems Software

Computers are increasingly becoming the natural delivery medium for teaching software packages (Shute and Region 1990). Kernighan and Lesk (1979) provided a tutorial environment on UNIX with controlled embedded access to the application software. Learners are allowed to manipulate the application software at certain stages during instruction. The tutorial was designed to teach the use of the operating systems. More recent application packages such as Lotus Smart Suite, and Microsoft

Office, have been released with built-in tutorial assistance. These tutorials allow the learner to interact with what appears to be the genuine application software. The tutorials allow for a subset of the operations that can normally be available making learner control difficult.

Numerous research papers have been published relating to ITS software. Most of these papers are very specific in terms of application, curriculum design, knowledge representation formalism, and method of delivery. Some of the publications include:

i. *Effect of Error Information in Tutorial Documentation.* This study is based on providing learners with error information in tutorial documentation (Lazonder and Meij, 1994). It is based on the hypothesis that users who used a manual with error information would develop better procedural skills than subjects who used a manual without error information. Forty-two subjects were randomly assigned to one of the two conditions. The result of the experiment shows that error information has no effect on procedural skills, both constructive and corrective. Although the aim of the experiment was clearly stated, the learners preferred learning strategy was not considered. The text system was interactive and is manual based. The result of the text may have been different if the software was context sensitive and a graphical user interface was used.

ii. *Socratic Tutoring.* Carbonell's (1970) research is concerned with enabling systems to engage in Socratic dialogs, believed to involve the learner more actively in the learning process, which may enhance learning. Collins (1977) outlined a set of tutorial rules for Socratic tutoring that were incorporated in the WHY (Stevens and Collins, 1977) system. It executes, for example, *IF* the student gives an explanation of one or more factors that are not sufficient, *THEN* formulate a general rule for asserting that the given factors are sufficient, and ask the student if the rule is true (Collins, 1977). Instead of semantic nets, the domain knowledge (e.g., flowering plan) was stored in a "script hierarchy" where information was contained about stereotypical sequences of events.

iii. *Expert Systems Tutors.* MYCIN (Shortliffe, 1976) was a rule-based expert system for diagnosing certain infectious diseases such as meningitis. GUIDON (Clancey, 1979) was constructed to interface with MYCIN for tutoring, and interactively presenting the rules in the knowledge base to a student. The way this tutoring occurred was as follows. GUIDON presented case dialogs where a sick patient was described to the student in general terms. The student had to adopt the role of a physician and ask for information that may be relevant to the case. GUIDON compared the student's questions with those that MYCIN would have asked and then responded accordingly.

iv. *Reactive Learning Environments.* Reactive learning environments allow the system to respond to learners' actions interactively. An early example of this kind of environment was SOPHIE (Sophisticated Instructional Environment), designed to assist learners in developing electronic troubleshooting skills (Brown and Burton, 1975), (Brown et al., 1982). In SOPHIE I, learners located faults in a broken piece of equipment by interactively communicating with the system. SOPHIE I included three main components: a mathematical simulation, a program to understand a subset of natural language, and routines to set up contexts, keep history lists, and so on. When troubleshooting a simulated piece of equipment, the student could offer a hypothesis about what was wrong. SOPHIE II (Lesgold et al., 1993), (Katz et al., 1993) extended the architecture of its predecessor by adding an expert based on a pre-stored decision tree for troubleshooting the power supply. Sherlock II provides a student with explanations/hints when there is an impasse. Finally, SOPHIE III represented a significant advance beyond the earlier versions in that it contained an underlying expert based on a causal model rather than a mathematical simulation. The importance of this change is that, in the original version of SOPHIE I, the simulator worked out a set of equations rather than by human-like, causal reasoning so it wasn't possible for the system to explain its decision in any detail. But the later version of the tutor (SOPHIE III) did employ a causal model of circuits to deal with the deficiency. Learners can communicate with the system by using a natural language dialogue mechanism and provide a critique for the learners. This system does not use a graphical user interface system to deliver instruction. The most essential feature of the system is the use of simulation as a workbench for learning to run its own experiments. It is interactive and thereby encourages learner involvement, and hence enhances retention and skill acquisition. The main problem with this system is that knowledge outside this domain cannot be taught, thereby limiting flexibility and robustness.

v. *LISP ITS.* Corbett and Anderson's (1992) description of the LISP ITS (LISP Intelligent Tutoring System) teaches students in the lisp programming language. It is based on theoretical principles derived from a theory of cognition proposed by Anderson known as ACT* (Anderson, 1990). ACT* theory proposes that human problem solving is enabled by a set of production rules, and can be turned into a formal set of well-ordered rules about instruction. Instruction based on the ACT* model should guide the learner through repeated practice, in order to facilitate procedural skill acquisition. Although, the ACT* model, does not directly address "buggy" and students misconceptions (Wilson and Cole, 1992). Based on a cognitive analysis of the domain, a representation of the student "buggy" and performance model has been implemented in LISP ITS. LISP ITS uses the finest grain size (Greer, 1992) as it analyses individual characters that students enter. It provides immediate

feedback after errors are made. The tutor's behaviour is linked to the student model, which consists of over 1200 rules. The tutor predicts the steps that the student might take in solving the problem, and with a known set of misconceptions, it models student errors at each step.

vi. *SOLA. Student On-line Advisor.* SOLA (Arshad and Kelleher, 1993) is an educational advice system that guides the student on what to learn and offers choice of educational materials. The system was implemented in SMALTALK-80, it involves determining study topics for the learner, and advising with appropriate teaching materials, when it should learn it. It is an advisory system, therefore it does not teach the specified course. Advisory systems are not suitable for an experience learner or mature student as they have formed their own learning strategy. SOLA should incorporate learning guidelines and control structures to facilitate the co-ordination and flow of information, and adapt different strategies for navigation to suit different learners.

vii. *MATHPERT.* Beeson (1990) has described MATHPERT (MATH EXPERT), an expert system in mathematics with extensive capabilities, supporting learning in algebra, trigonometry and introductory calculus. It provides step-by-step solutions, tailoring teaching methods and step size to individual users through student modelling. MATHPERT does not allow students to make mistakes, thereby eliminating the "buggy model". Self (1990) proposed an ITS without a student modeller as a means of "bypassing the intractable problem of student modelling". Self (1990) argues that the role of student modeller is essentially to analyse student errors and misconceptions. It can be inferred that misconceptions and mistakes during instruction are corrected, and this negates the need for a student model.

viii. *DOMINE.* Domine system, (Spensley and Cook, 1988) has a knowledge elicitation phase that captures static screen dumps from the application. These can then be displayed to the learner as an appropriate teaching operation. It does not allow the learner to interact directly with the software being taught.

ix. *Kimball's Integration Tutor.* Kimball's integration tutor (Kimball, 1973) was developed to teach integration. The teaching method employed is interventionist, when the learner is in trouble, the program intervenes by employing a dialogue rule. The learner carries out integration on a computer terminal and the program checks each transformation to see if the learner has applied integration principles correctly. The program uses artificial an intelligence method for integration and algebraic simplification. Using these two techniques, Kimball's tutor can solve symbolic integration problems. The program also has a lot of problems and solutions, and will select examples for students when requested.

x. *PROUST.* PROUST (Soloway, et al. 1991) is a tutorial system for teaching Pascal. PROUST attempts to identify and report possible errors during compilation of program statement. It is a non-interactive tutorial system. It works by understanding what the learner wants to do and how they are going to do it.

xi. SHERLOCK. SHERLOCK (Lesgold et al., 1993), (Gott et al., 1996) developed for teaching complex electronic troubleshooting job. The system provides expert explanation to troubleshooting problems. It also helps students compare the systems' solutions with their own solutions at the end of each problem solving task

xii. *LISP.* This is a teaching program for teaching programming language. It is based on adaptive control of thought (Anderson and Reiser, 1985) within the interactive environment of the LISP interpreter. Errors are detected and reported immediately after they are committed. The LISP approach can be applied to a wide range of software interfaces.

xiii. *Leeds Arithmetic's Teaching Programs.* Leeds Arithmetic's Teaching Program (Woods and Harley, 1971) was developed to teach arithmetic and to generate and administer exercises in arithmetic problems at diverse levels of difficulties. The system is context sensitive and can provide materials suitable to the learner level of competence, it also provides different types of feedback and error information. The major shortcoming of the system is that the learner cannot interact directly with the system, use of a GUI will greatly improve the user interface.

xiv. CATO (Aleven and Ashley, 1997) helps students building legal arguments by generating relevant case scenarios. The system also reifies the connection between the content of the cases and their use in the arguments.

xv. DISCOVER (Ramadhan, 1992) is a learning environment for a very simple algorithm-like pseudo-code language. DISCOVER, following Du Boulay et al.'s (1981) glass-box approach, is designed to offer students an opportunity to "look inside" the virtual machine, i.e., to see how the values of variables change when a program executes. To achieve the goal of making intrinsic activities of program execution explicit to students, DISCOVER provides a window with an example solution and a window showing the current values of all variables. This approach is similar to Mayer's (1981) approach of supporting "concrete models" of computers and software systems. Although, DISCOVER seems to be a good learning environment for learning to solve programming problems, it deals with a very simplistic programming domain and does not support iterative or recursive programming.

Viz (Eisenstadt et al., 1992) works in the same domain as PROUST, but it supports a software visualisation technique instead of program code analysis for understanding students' programs. Software visualisation techniques help programmers see what their programs do using meaningful graphical abstractions and, as a result, programmers learn better programming

techniques for constructing programs in the first place. Viz provides tools to view "monitorable program constructs," and tools to view snapshots of executions of algorithms.

Most of these ITS software applications are domain specific and their functionalties are limited to the development platforms. Some of this research also discourages attributions by providing a detailed model of how new domains can arise only from the current developments. Furthermore, many ITS authoring tools sacrifice pedagogical requirements for content flexibility (Bell and Zirkel, 1997). Table 1A summarises reviewed ITS research publications, indicating different emphases placed on different ITS components development.

Table 1A ITS Research Publications

AUTHOR/YEAR	DESCRIPTION
Carbonell (1970)	SCHOLAR system, geography tutor, based on semantic tutor.
Brown et al. (1978)	DEBUGGY, attempted to identify and correct bugs during instruction.
Burton and Brown, (1978)	Buggy, Subtraction, procedural network of tasks.
Burton and Brown (1979b)	WEST, Arithmetic tutor.
Brown et al. (1982)	SOPHIE, electrical trouble shooting.
Hartley and Sleeman (1973) Sleeman and Brown (1982)	ITS must possess: (i) knowledge of the domain (expert model), (ii) knowledge of the learner (student model), and (iii) knowledge of teaching strategies (tutor).
Goldstein (1982)	WUSOR, teaches logic relations based on genetic graph network.
Millar (1982)	Logo programming tutor in SPADE.
Sleeman (1982)	Algebraic procedures, in LMS.
O'Shea (1982)	QUADRATIC, quadratic equation tutor.
Clancey (1982)	GUIDON, infectious diseases
Soloway et al. (1983)	PROUST, programming in Pascal.
Lantz et al. (1983)	Applied ALGEBRA.
Soloway et al. (1991)	SODA: adaptive learning environment; provides scaffolding knowledge support.
Eisenstadt et al.(1992)	Viz provides a software visualisation technique for programmers.
Ramadhan (1992)	DISCOVER is a learning environment for a very simple algorithm-like pseudo-code language.
Anderson (1993)	Anderson lists of errors are embedded in specific production. Manages all interactions between the student and tutor.

Some of these research publications use expert systems and case-based reasoning methods for capturing and representing domain knowledge. These publication illustrate the many trade-offs between conflicting "soft" research goals such as domain specification, interactivity, run-time efficiency, modularity, flexibility, reuse of pre-existing components, and production of reusable

components. For example, the researcher may have placed higher priority on clear conformity to the domain specification than on easy-to-understand instructional modules and reusability. ITS development requires a "conceptual shift" from traditional approaches to a more reusable and modular knowledge representation (Murray, 1996).

APPENDIX B

BAYESIAN NETWORK

B1.0 Introduction

In a complex environment, no matter how good the instructional environment is designed, it will sometimes be unable to make adequate predictions about the learner's activities and behaviour with definite certainty. Uncertainty may rise from the environment, from the user's activities, and from the inference systems and learners path through instruction. In some case, the student model may be inadequate. Some major sources of uncertainty (Russell and Norvig, 1995) include:

i. *Randomness*. The environment may be un-deterministic. In this case, an omniscient observer would be unable to predict the world correctly.
ii. *Complexity*. There may be a number of unlikely exceptions that would be expensive to enumerate, or the "true" theory of the world would take too long, or be too large, to learn precisely.
iii. *Representational Limitations*. The system may be unable to express a correct deterministic theory (event if one exists) in its concept language.
iv. *Interactive Limitation*. Interactive feedback only reports a limited number of information about the learning activities. If important data are not reported, the system may be unable to acquire the knowledge necessary to characterise the world precisely.
v. *Feedback Inconsistency*. Feedback may always report the same perceived world in identical world states, due to noise in the sensors.
vi. *Missing data*: this may be a result of incomplete representation/acquisition during analysis and development. The causes a "gap" in the different sources of knowledge used.
vii. *Invalid data* may result from input errors and specification errors or knowledge.
viii. *Relevant*: Training instances may not be relevant (e.g., Mitchell, 1983) due to classification methods.

These sources of uncertainty are related. For example, an environmental factor may greatly influence the activities of a user and the effect is passed to other instructional activities.

Furthermore, a distorted model may result from using incorrect inferences. Similarly, if the environment is deterministic, in such as a way that outcomes are heavily dependent on initial conditions, then incorrect inferences may result due to insufficient theory or noise.

B1.1.0 Evaluating Probabilities Theories

This thesis developed a Bayesian method for estimating the quality of "theories". The essential requirements are: the accuracy of the theory, given by the likelihood of evidence $P(E|T)$ and the prior probability of the theory $P(T)$. The former is computed using probabilistic combination independence (PCI).

In most models of concept learning, observations are assumed to be correct (i.e., noise free) and consistent with some deterministic hypothesis that is expressed in the hypothesis language (for example, (Mitchell and Keller, 1983), (Kuipers, 1985), (Carbonell and Gil, 1987). The concept-learning problem under this assumption becomes that of finding a hypothesis in the concept space, which is consistent with all of the observed instances of the concept to be learned. A consistent hypothesis may be prioritised according to some preference metric (e.g., of simplicity or specificity), or they may all be considered equally good (as in the version space algorithm).

In a non-deterministic or noisy environment, we can no longer expect to find a completely consistent hypothesis. The problem then becomes that of finding the hypothesis that "best describes" the observed instances.

The question is how to define "best description?" If we allow enough parameters in the concept description, there will always be some theory that is consistent with all of the data, for example, we can just take the disjunction of all of the observations. The problem with this approach is that the resulting theory will be cumbersome and expensive to use, and is not likely to make any useful predictions (or perhaps not able to make any predictions at all).

Using a simpler theory has two advantages. First, minimising error on the training set may actually cause the theory to be fitted loosely and therefore will not minimise future error. Second, if the simpler theory is less accurate, the cost saved in applying it may outweigh the loss of accuracy for a limited rational agent (able to make a limited number of deductions in a limited time). The approach adapted in this thesis is based on Bayesian probability theory.

B1.1.1 Theory Notation

The short-hand notation for theories is used through the rest of this section. Rules are represented as implications with attached probabilities. They should not be interpreted as logical implications, but as conditional probabilities, for example.

$$action(t_1 : elicitate) \rightarrow_{0.6} \Delta u(t+1,90)$$

Represents the conditional probability.

$$p(\Delta u 9t = 1,90) | action(t_i elicitate)) = 0.6$$

The symbol is used to indicate an empty conditioning context in a single rule. Represented conditional contexts in a single rule are left out for readability.

B1.1.2 Bayesian Knowledge Representation

In order to use probabilistic knowledge in an automated knowledge acquisition module, a formal system for representing and reasoning with probabilities is required. Bayesian knowledge representation (Eugene and Bank, 1995) is used to represent the domain knowledge. This approach reduces oversimplification of the knowledge variable by training their "independence assumptions in terms of conditional dependency" (Eugene and Bank, 1995). This approach is widely used for military applications where the source and application of the domain knowledge is crucial. Bacchus' (1990) probabilistic logic and Pearl's (1988a), (Pearl, 1993) belief networks provide formalism for representing probabilistic knowledge. Each of these approaches are discussed below:

Logic and Probability: Bacchus' probabilistic logic is a formal language for representing probabilistic knowledge using first-order logic. The language provides a representation for both statistical probabilities (which may be defined in terms of observed frequencies of events) and subjective probabilities (degrees of believe derived from the statistical probabilities). The inference mechanism provides for some manipulation of the statistical probabilities using standard axioms of probability, and for direct inference from statistical to subjective probabilities using the narrowness reference class.

A direct inference from statistical to subjective probabilities is based on finding a statistical probability with the same reference class as the desired subjective probability. If there is no probability, a simple type of independence is assumed non-monotonically (i.e. until additional evidence shows otherwise), and the "next-narrowness" reference class for which a probability is

available is used. Bacchus' approach provides a useful formalism for representing many aspects of probabilistic reasoning, including certain forms of default reassigning. However it does not provide a representation for beliefs about relevance, nor does it allow default assumptions such as independence to be used in the inference process.

B1.2 Bayesian Network

A Bayesian network (BN) is a directed acyclic graph in which nodes represent variables of interest, and edges represent associations among these variables (Pearl, 1988a). To quantify the strengths of these associations, each node has a conditional-probability table that captures the relationships among that node and its parents.

Bayesian-network semantics are founded on the principle of *conditional independence,* which dictates that if the states of a variable's parents are known, all other information adds nothing to our attempt to predict that variable's states.

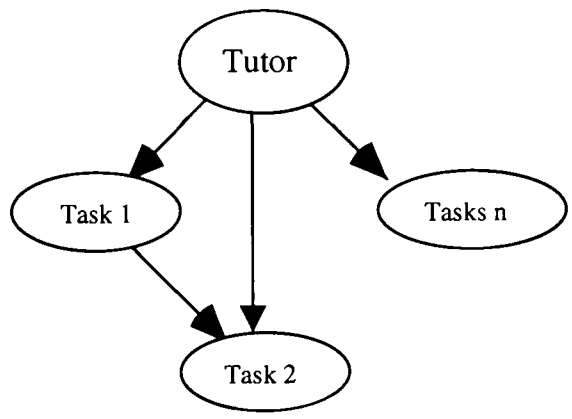


Figure B.1 Example of a Bayesian Network for a Tutor

As illustrated in Figure B.1, each object and their attributes have a unique identifier that specifies their class, and domains. Figure B.1 depicts an example of a Bayesian network for a Tutor, with Task 1, and Task 2, which are mutually exclusive and represented as variables. Each class is represented based on their probabilistic information and their attributes as described above. The Bayesian network provides a mechanism for computing and analysing user actions by using a combination of the following techniques:

i. identifying different classes of components and their attributes;
ii. identifying the constraints imposed by these classes;
iii. abstracting the component/module from the class schema;

iv. dynamically generating explanations in response to user queries and feedback obtained from the student model;

v. inferences are made from background knowledge and tutorial exploration.

Some of these techniques are discussed further in this section.

B1.2.1 Representation

The GeNisa knowledge base uses probabilistic theories about the domain, i.e., instructional goal/task to make inferences. Each theory consists of a set of conditional probabilistic distribution of values of a specific task. The learning task is the area to where knowledge is to be acquired, i.e., the conditional context. A probabilistic inference representation is used to make predictions (assumptions) about the effect of each attribute and the learning goal. The mechanism requires the determination of which conditional distributions within a theory are relevant and combining them if necessary (using minimal independence assumption (discussed later in this chapter) to get a single predicted distribution.

B1.2.2 Conditional Probability

Definition. The conditional probability (CP) of X given Y is X, the target, and Y, the conditioning context (CC), are first-order schematas. These features are required to be conjunctions of feature specifications, where each node in a feature specification may contain internal value disjunction representing internal nodes at time t. For example the following task is valid.

$$(t, tutorial[1,30])$$

This implies that, at time t, the learner was using tutorial (units) between nodes 1 and 3. This schema corresponds to a set of perceived actions. The following task is valid.

$$tutorial(t, apptask, 1) \wedge case(t, [10, \infty]) \wedge \Delta u(t, -100$$

Note negation of features is not allowed, since they may be written as a disjunction.

An example of a condition probability is

$$P(\Delta u(t+1), -10) \,|\, action(t, : move - forward)) = 0.75$$

The variable t, represents time at which the conditional probability is the most specific in the current domain, i.e., the knowledge about the task at time t, implies CC and does not imply any other more specific CC.

Variables are universally quantified, since they cannot be instantiated without the rest of the theory. The semantics of any individual probability theory will depend on the context of the rest of the theory as well as on the inference mechanism used to instantiated the variables and make predictions. Using conditioning context and relevance in this way yield a "quasi-non-momtonic" representation, which adds new knowledge, i.e., new conditional probabilities to a theory doesn't change the rest of the CP in the theory, but it may change the range of applicability and semantics.

B1.3 Conditional Distributions

Definition: A conditional distribution (CD), which is henceforth referred to as rule, is a set of n conditional probabilities on a target schema G (a learning goal), with mutually exclusive partial variable substitutions $\theta_1 \ldots \theta_n$ and common conditioning context C, such that

$$\sum_{i=1}^{n} P(G\theta_1 \mid C) = 1$$

A CD specifies all of the possible instantiations for a target given a particular context, and their probabilities. If C contains all of the relevant information, this distribution is used to predict the probability of each value of G. A CD of an empty conditioning context is referred to as a prior distribution on the given domain task. A prediction on G is a set of probabilistic outcomes specified by a conditioning distribution.

B1.3.1 Predictive Theories

Definition. A predictive theory (PT) on goal schema G is a set of m conditional distributions, or rules, on G, with conditioning contexts $C_1 \ldots C_m$ (which must be distinct but not necessary disjunction), such that any situation (consisting of a perceived world and possibly an action) implies at least one of the conditioning contexts.

As long as a set of distinct rules on a goal schema includes a default rule it is guaranteed to be a predicative theory.

A predictive theory stores all of the beliefs about the current goal G. The rules in a theory are indexed by their conditioning contexts (i.e., the situations in which they apply). Using a specificity relation between CCs, the rules can be organised in a directed acyclic graph (DAG) in which a child is always more specific than its parents.

Figure B.2 below shows an example of a predictive theory on a goal G, drawn as DAG. Only the conditioning contexts are shown, indicating the structure of the theory. A conditional distribution is stored at each node. For example, the bottom node represents a rule containing conditional probabilities of the form

$$P(G\theta_i \mid A(x) \wedge B(x)) = p_i$$

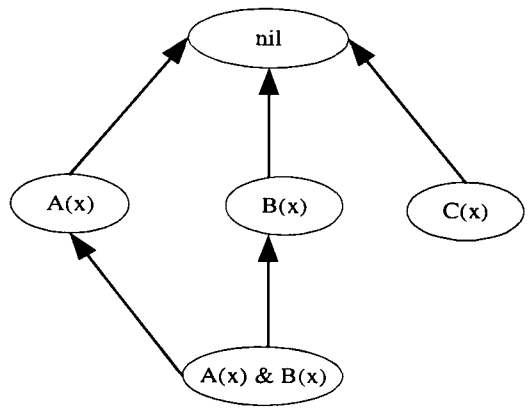


Figure B.2 Example of a Predicative Theory

The above procedures can then be used to identify valid independence assumptions and use them to find the joint distribution The independence assumption technique involves finding a shared feature in the conditioning contexts of rules to be combined and assuming that the remaining features are independent, given feature.

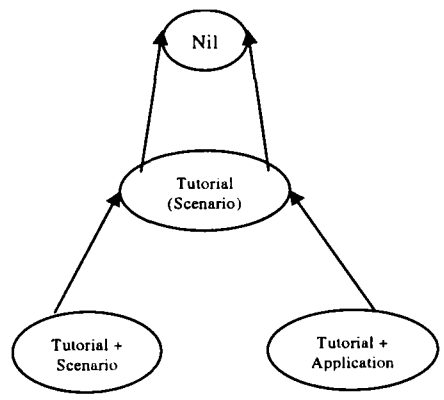


Figure B.3 Example of a Predictive Theory

Definition. The shared feature of a set of rules is the feature that appears in the entire conditioning context and has some attributes in common. "Share feature" may also refer to the share sets of values for the attributes of the current class/domain. For example, the share feature of a tutorial class:

$tutorial(x) \wedge task-unit(x)$ & $Sceanrio(x) \wedge Unit(x)$, (Unit (x)) is the shared feature.

B1.4 Probabilistic Inference

Probabilistic inference uses independence assumptions described earlier in this section. However, prediction about the environment can not be inferred due to uncertainty. In general, it will be impossible to be certain about the sources of uncertainty. The system cannot know a priori which uncertainty sources are present.

There are functional advantages for using a probabilistic representation of theories. (i) Probabilistic representation are less brittle than deterministic theories, i.e., the behaviour of the system degrades as the quality of the theory decreases. (ii) Statistical probabilities represents summaries of observed frequencies of an event, i.e., learning relevant "events".

Subjective probabilities are used to evaluate the domains predictive theories. In order to decide which theories are most effective, a Bayesian analysis is performed with the likelihood of the evidence (computed using the statistical probabilities in the theories).

Theory Evaluation. This research developed Bayesian methods for measuring the quality of "theories". The essential requirements are: (i) the accuracy of the theory given by the likelihood of evidence *P (T)*, and (ii) the prior probability of the theory *P(T)*, (iii) a priori preferences (i.e., learning biases) can be expressed as prior probabilities, which are gradually overridden by data. The former quantity is computer-generated using probability conditional inference (defined because of simplicity).

B1.4.1 Discourse Planning

The discourse planner performs a random action over a fixed percentage of time (default probability 0.25). The remainder of the time it uses a heuristic search to find the action with maximum overall expected utility. The overall expected utility is equal to the immediate expected utility of performing the action, plus the maximum utility of the action plan that can be formed in the resulting states.

The discourse planner forwards chains to a fixed depth (default 3) through the space of possible outcomes for each action, then propagates the maximum expected utility backwards to yield an expected utility for each initial action. An example of part of the discourse process is shown in the Figure B.4. Only the left half of the plan is fully expanded. The theory used is:

$$\rightarrow_{0.7} \Delta(t+1,-10)$$

$$\rightarrow_{0.3} \Delta(t+1,-11)$$

$$action(t, : elicitate) \rightarrow_{.5} \Delta(t+1,90)$$

$$\rightarrow_{0.5} \Delta(t+1,-10)$$

$$action(t, elicitate) \wedge \Delta u(t,90) \rightarrow_{0.67} \Delta u(t+1,90)$$

$$\rightarrow_{0.33} \Delta u(t+1,-10)$$

The square represents predicted changes in utility; the capsules contain the expected utility of the entire discourse plan. The expected utility in the bottom row is computed directly using probabilities to weight the predicted utilities. For example, the leftmost expected utility is equal to (0.67*90 + 33*(–10)). The values in the upper row of capsules are computed by taking the maximum expected utility of the rest of the plan from each state, plus the immediate utility of the state and by weighting the probability of the state. For example, :elicitate has the highest expected utility in both of the lower states, so the expected utility of performing :elicitate (57 and 40, respectively for the two possible outcomes) is propagated backwards. The overall expected utility of performing :elicitate as the first action (represented by the upper left capsule) is then (90 + 570 *.5 + (–10 + 40) * 5, or 88.5. The overall expected utility of :move-forward is 29.7 , so the discourse planner selects :elicitate.

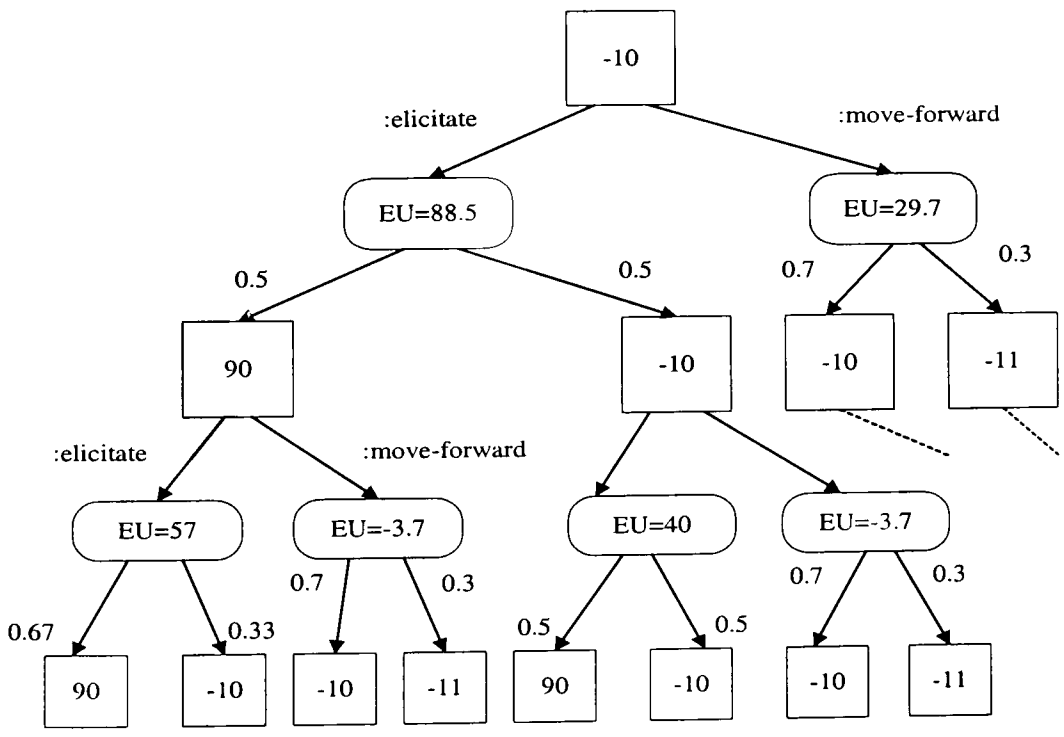


Figure B.4 Partial Discourse Plan Tree

B1.4.2 Task Directed

The discourse planning uses principles of decision theory to choose discourse units. The expected utility of being able to compute the various utility of the environment is computed and those with the highest utility are used during discourse. As user action progresses, GeNisa uses the learning theory to determine which feature of the scenario, i.e., learning features, it expects to be most useful to learn next.

> *Definition*: A unit step treats the difference in two plans (actions) generated by the user using different scenario models as the differences in the final step of each plan.

For each learning goal, the user uses background knowledge to select discourse and induces a predictive theory for the goal from observations of the domain action. The utility of the intermediate steps is assumed because the utility of the intermediate steps varies.

B1.4.3 Knowledge Acquisition

The knowledge acquisition module is based on probabilistic learning. Knowledge bases are generally characterised by uncertainty caused by:

Representation. As discussed earlier, incomplete/incorrect inferences may have been drawn during analysis and development. For example, data may be eliminated during analysis and development (e.g., (Dirtherich, 1982), (Mitchell, 1983)) where the training cases were pre-classified into relevant instances and non-instances. There may also be a "gap" between the different sources of knowledge.

UnreliableData. The sources of discrepancies may result from expert views, i.e., discrepancies in experts' knowledge and methods of representation.

In general, it will be impossible to be certain what the sources of uncertainty are. The system cannot know a priori which uncertainty sources are present.

Subjective probabilities are used to evaluate the users' predictive theories. In order to decide which theories are most effective, a Bayesian analysis is performed combining a prior probability with the likelihood of the evidence (computed using statistical probabilities in the theories).

B1.5 Bayesian Probability

The theory with the highest probability should be that with the most effective structure for representing the observed data. Given this structure, the probabilities within the theory are straightforward to optimise. Complex structures (those with many dependencies) take too much computation time and are characterised by a risk of "over-fitting". On the other hand, a simple structure with only a few dependencies may not capture important relationships in the world.

The probabilities we wish to find are the probabilities that the structure of this theory is the best representation of the behaviour of the environment. It is not in the probability that the particular values of the conditional (statistical) probabilities are estimated using observed frequencies; this maximises the accuracy of the theory as given by the Bayesian likelihood $P(E|T\cdots K)$. Using the notation:

T: A proposed theory

K: Background knowledge

E: Evidence: a sequence of observation $e_1, e_2, \ldots, e_n$

the Bayesian rule gives:

$$P(T \mid K \wedge E) = \frac{P(T \mid K)P(E \mid T \wedge K)}{P(E \mid E)}$$

We are only interested in finding a relative probability in order to compare probabilities of competing theories, so the normalising factor $P(E|K)$ in the denominator can be dropped yielding

$$P(T \mid K \wedge E) \alpha P(T \mid K)P(E \mid T \wedge K).$$

We also assume that the individual observations $e_1 \ldots e_n$ composing E are independent, given K and T. This standard conditional independence assumption is reasonable because the theories generated by the user make independent predictions. Therefore, T embodies an assumption that the observations are independent, which must be true if T holds. Therefore, the first quantity on the right-hand represents the "informed prior", i.e., the probability of the theory given the background knowledge K, but no direct evidence. The second quantity represents the likelihood of the theory, i.e., the combined probabilities of each piece of evidence given the theory and K.

B1.5.1 Prior Probability

The prior probability of a theory, $P(T)$, is the probability of T before any evidence has been collected. A "prior", however, is never completely uninformed an event before any direct observations about a particular leaning task are made, a user's past experience, available sensors, and internal representation will affect its disposition to believe a theory, and hence its prior probability distribution. For example, even if you have never been to a particular theatre, your general background knowledge about theatres allows you to learn quickly how to buy tickets and refreshments, and how to find your seat. All of the background knowledge available to the user should ideally be reflected in its "prior".

$$P(T \mid K \wedge E) \alpha P(T \mid K) \prod_{t=1}^{n} P(e_t \mid T \wedge K)$$

Encoding should take into account not just the probabilities of individual terms, but the probabilities of pairs of terms. In some domains, however, even this will not be enough: structure in the language may affect large groups of terms. Individual terms may not be the right level to consider for computing "prior", rather, higher level classification should be used to evaluate a theory (Good, 1983). These categories might be semantic groupings of similar words or ontologies.

B1.5.2 Uniform Distribution of Theories

Uniform distribution allows equal probabilities to be assigned to every theory. An infinite theory space may result in improper prior (i.e. all theories have zero prior probabilities), but since we are interested only in relative probabilities, we can ignore the prior probability terms and simply choose the theory with maximum Bayesian likelihood P(E|T). This procedure finds a theory that exactly fits the data, if one exists. In cases of a tie (where two theories have equal likelihood) the shorter one will still be preferred (i.e., the theory with fewer rules r, if the theories being compared have the same number of rules, the theory with fewer terms).

Suppose P_R has constructed the following two simple theories, T_1 and T_2.

$T1 \rightarrow 0.5\, \Delta u(t+1, -10)$

$\rightarrow 0.5 \Delta(t+1, 90)$

$T2 \rightarrow 1.0\, \Delta u(t+1, -10)$

action(t,: elicitate) $\rightarrow 1.0 \Delta$u(t + 1, 90)

Suppose that the evidence used to construct these theories consists of two observations:

e_1 = action(1,: elicitate) $\wedge$ Δu(2,90)
e_2 = action(2,: move - forward) $\wedge$ Δu(3,-10)

$$\rightarrow 1.0\Delta u(t+1,10)$$

The uniform distribution on theories assigns the same probability to the two theories *($P(T_1)$ = $P(T_2)$)*. The likelihood of the two theories is simply the conditional probability of the evidence, given the theories:

$$action(t_1 : elicitate) \rightarrow_{1.0} \Delta u(t+1,90)$$

$$p(E \mid T_1) = p(e_1 \mid T_1)p(e_2 \mid T_1) = 1/2 * 1/2 = 1/4$$
$$p(E \mid T_2) = P(e_1 \mid T_2)p(e_2 \mid T_2) = 1 * 1 = 1$$

Applying the Bayesian evaluation formula:

$$p(T_1 \mid E) \ \alpha \ p(T1)p(E \mid T1) \ \alpha \ 1/4$$
$$p(T_2 \mid E) \ \alpha \ p(T_2)p(E \mid T_2) \ \alpha \ 1$$

Therefore, under the uniform distribution of theories, T_2 is preferable given the evidence.

APPENDIX C

EVALUATION CRITIERIA

C1.0 Introduction

This section provides materials used for both formal and informal evaluation of GeNisa. Evaluation materials were collated throughout this research.

C1.1 Evaluation Criteria

This section provides a brief description of evaluation criteria presented in Chapter 5.

In the content criteria section of the questionnaire, the participants rated the objectives, the difficulty level, the component modules and usability as appropriate. Generally, the participants ranked the entire questionnaire as accurate. Likert scale for first five section:

Strongly Agree	Agreed	Neutral	Disagree	Strongly Disagree
5	4	3	2	1

The means of effectiveness criteria rankings ranged from 4.0 to 4.83. The highest ranked criterion was that there was a short delay between student response and program feedback. The lowest ranking indicated that the users felt that the program did provide exit options, but that the participants only agreed with this item rather than strongly agreeing.

In the section regarding impact criteria, the highest ranked item was that incorrect responses were given feedback about why they were wrong, or hints to move them closer to a correct response (mean 5.0). The next highest ranking was that knowledge was connected sufficiently through hyperlinks (mean 4.83). The other items in this section were ranked with means ranging from 4.5 to 4.0. The lowest scored item was that simulation example met objectives and built intuition, although with a mean ranking of 4.0, the participants still agreed with this criteria.

The effective use of media was evaluated, with means ranging from 3.60 to 5.0 for the criteria. The participants strongly agreed that the graphics, animation and pictures were appropriately used.

Participants rated that GeNisa provides immediate and satisfying feedback and would hold their attention (means 5.0 for both items). The lowest rated item was that students would not need additional training to meet objectives (mean 3.83), and that documentation and instructions were adequate (mean 4.16). The overall opinion of the participants was that the courseware was adequate for instructional use (mean 4.83).
For the final section, Likert scale values are:

Difficult	Neutral	Easy
1	2	3

User interface ratings were generally very positive, with means ranging from 8.0 to 9.67 (on a ten-point scale). Highest ranked criteria were ease of use (mean of 9.67) and easy to begin program (mean 9.5).

The lowest ranked item was mapping (which indicates whether the program gives information about where the user is within this GeNisa) with a mean of 8.0.

Four quantitative criteria that the software was evaluated against are defined below:

i. *Enquiry.* This criteria refers to software design, screen display, logical operations, ease of movement and the variety and flexibility of search options. Other aspects considered include whether the screen was configurable for different users, ability to create, sort and use the graphical package and connection to the Internet or networks.
ii. *Collection Management.* This criterion assessed file management and general media management and editing of different functions. This criterion also considered the availability and quality of help messages, and ease of management modules
iii. *Data Import/Export.* Import and export of different classes and classes developed by other vendors. This criterion also measure data backups and data-processing modules
iv. *Online Support.* Considers content-sensitive help between modules and types of feedback provided.

C1.2 Qualitative Evaluation Criteria

i. Quantitative elements include configuration, help support, reliability of the software. performance failure, and usability.

ii. *Configuration.* Minimum configuration and was assessed in this section. Items included the operating system required, network type and platform requirements.

iii. *Help Support.* This includes the availability of prompts, and adequate and appropriate support given to different component modules

iv. *Reliability of the Software.* This measures the stability of the GeNisa in different operating environment. Here, a few users reported minor glitches, some bugs that could be worked around or problems associated with platforms and operating systems.

v. However, it was observed that there were no instances where these bugs caused any serious damage or interrupted the operation of the software and the operating environment. Problems were associated with differences in Java classes and were immediately resolved with new classes.

vi. *Usability.* Each component was assessed on whether it was easy or difficult to use. The degree of usability was evaluated in terms of user feedback and HCI guideline, and on whether the tools were intuitive and logical, especially the "search" and "case" tools. The main emphasis was placed on logical operation and use of files, classes management and editing functionality, lists to choose from, and default response incrementally during data entry.

This section provides sources of essential evaluation criteria used for the evaluation of GeNisa described in Chapter 5.

Table C 1. Evaluation Criteria

Criteria	Sources	Definition
Architecture	Literature and Development Experience	Modular architecture with different user interface for courseware authoring and delivery.
Courseware Development	Literature and Development Experience	The tool should allow development of instructional units visually, i.e. by using icons and graphics objects.
Learning Environment	Literature and Development Experience	The number of potential new software systems to be developed or existing software systems enhanced in the tools should support different use.
Reusability	Literature and Development Experience	The potential to identify reusable components in the domain and each component reusability.
Courseware Generation	Literature and Heuristics	The courseware may be generated automatically form users activities and from case scenarios.
Components Representation	Literature and Development Experience	The size of the scoped domain versus the available resources and expertise to complete the effort.
Unique Features and	Literature and	The amount of knowledge about reuse possessed by the

Criteria	Sources	Definition
Functionality	Development Experience	domain experts including knowledge of tool.
Algorithms	Literature and Heuristics	For embedded controls and to enhance application functionality.
Stability	Literature and Development Experience	The degree of change to the functionality to the components within during the tool enhancement.
Dialogue	Development Experience	Flexible, mixed-initiative dialogue.
User Support	Development Experience	This is concerned with the perceived facility with which a user interacts with an interactive application program.
Domain Expertise	Literature and Development Experience	The domain experts should provide support for the learning environment, and should have adequate domain expertise and tools to support instruction.
Instructional Content	Literature and Development Experience	The ability of domain experts to provide adequate instructional content and to assist within a knowledge acquisition. Ability to review instructional contents, and have flexibility for managing the tutorial discourse, and according to instructional strategies.
Learner Monitoring	Literature and Development Experience	The programs ability to track and graphically represent to the user his or her path through the program; provides content domain specific feedback to user performance.
Case Scenario	Literature and Development Experience	The availability of usable domain scenarios, to assist instructional activities. Scenario library of different cases and use of realistic case scenarios.
Scalability	Literature and Development Experience	The expected consistency of all components functionality; and the ability of the system to withstand additional tools, users, processes and data with minimal impact upon the overall performance.
Portability	Literature and Development Experience	Platform independence, usability, including code, algorithms, etc. The use of widely accepted and implemented computing standards.
Development Methodology	Literature	Whether a consistent development methodology for applications has been employed.
Graphic and Multimedia Design	Literature and Development Experience	Do the visual effects enhance the resource, distract from the content? If audio, video, virtual reality modelling, etc are used, are they appropriate to the purpose of the source?
Interactivity	Literature and Development Experience	This includes screen design (relates to text, icons, graphics, colour, and other visual aspects of the programs); navigation, i.e. ability to move through an instructional contents in an interactively; associative recall (use of effective cueing methods), provision of meaningful error messages; reduce short-term memory load where possible.
Connectivity	heuristics	Can the resource be accessed with standard user interface, or network requirements? Can the resource be accessed reliably?
Scope	Literature and Development Experience	Does the actual scope of the components match your expectations and all area of use covered?
Audience	Users trial	The target recipients of the system such as learner/developer. Quality of the instruction/tools provided.

Criteria	Sources	Definition
Issues	Literature and Development Experience	The use of appropriate varieties of instructional principles, which may facilitate acquisition and retention of tutorial contents.
Flexibility	Development Experience and Users trial	The ability of the systems to be configured in a variety of ways and across platforms.
Instruction	Development Experience and Users trial	The system should be easy to learn. The lessons should be structured with particular instructional steps or sequences directed to particular domain task.
Efficiency	Development Experience and Users trial	The system should be efficient to use, with a resultant high level of productivity.
Cognitive Load	Literature and Development Experience	The system should be intuitive and be easy to remember. Should minimise cognitive load on the user (e.g., Shneiderman, 1998). The user interface should reduce short-term memory load as much as possible.
Associative Recall	Literature	Components should provide cues that cause users to recall related events or tasks e.g. effective cueing.
Problem Solving	Literature and Development Experience	Realistic problem solving, with adaptive responses and the use of graphical illustrations as instructional aids.
Knowledge Acquisition	Literature, and Heuristics	Automated, and mixed initiative knowledge acquisition method.
Database Support	Literature and Development Experience	Ability to store application specific data represented in a formalism that allows cross platform utilisation and reuse.
Cost	Literature and Development Experience	Costs associated with the development, use of the system and instructional content delivered.
Errors Tolerance	Literature and Development Experience	The system should have a low error rate, and allow users to recover from mistakes.

C2 General Features and Rationale for Evaluation Criteria

This section presents the rationale for using evaluation criteria in Table C1 and a general features to be considered for evaluating an ITS. A number of the criteria in Table C1 may be combined to meet application specific criteria or platform requirements. The general features are listed below.

i. *Domain Specific.* Relates to domain specific content required for effective application development and delivery. The identification of the trade-offs between conflicting "soft" goals such as clear conformity to the specification, easy-to-understand code, run-time efficiency, modularity, modifiability, flexibility, and reuse of pre-existing components.

ii. *Generic Features.* Identification of features that could leveraged future development. This also involves the use of existing components that may satisfy the current requirements, or that may be instantiated for specific tools. Developed components should support short and long-term objectives.

iii. *Integration.* Ability to share data across platforms and in a language neutral way is essential for developing a cohesive system. The system must support backward compatibility in order for it to be easily evolved and allow developers to modify the

existing tools. This may increase the flexibility of the application development process and support future enhancements.

iv. *User Interface.* The user interface must provide a seamless way to navigate between the different components in a consistent manner. The user interface must provide support for seamless integration with other tools. Each user interface tool must be clearly visible, recognisable, and contain an icon represented on the standard toolbar. Users activities must be specified and invoked by direct manipulation of graphical representation of the tool (Shneiderman, 1998).

v. *Forcing Function.* The dialogues implemented by the user interface should either be non-exclusive or exclusive. A non-exclusive dialogue, allows users to interact with a tool and invoke another tool without having to close the first dialogue. Examples of this are dialogues that display a message to the user or an option window. Non-exclusive or exclusive dialogues could be used for forcing functions. A forcing function (Norma, 1988) prevents a user from performing actions that are not necessary in a given context. For example, a user should not be allowed to proceed to the next tutorial tasks until all prerequisite tasks, including case scenarios have been completed.

vi. *On-line-Help.* On-line-help should be provided for all components. The on-line-help should be goal-oriented, descriptive, procedural, interpretive and navigational (Sellen and Nicol, 1990).

vii. *Reduced Working Memory.* By using associative recall (Ellis and Hunt, 1993) for all tools and user activities. This may also be implemented by using interactive strategies that minimises cognitive load and by providing subgoals, plans and providing hints that may assist with the completion of the task.

viii. *Application Requirements.* Matching the application requirements with the tools provided by the system. The system should provide means by which other tools can be easily prototyped. Also, there must be a clear separation between components and their underlying events. This allows flexibility by proving permitting reuse of existing tools and development of new ones.

ix. *Instructional Content.* The instructional contents are hierarchically organised and easy it is easy to make enhancements and modifications. It should also support different learn levels of activities.

x. *Level of Expertise.* Level of expertise required for using the components and the easy to which the tools can easily be integrated into new development must by minimal.

xi. Demonstration of unique functionality associated with different components.

xii. *Platform Specific Features*. This relates to consistency of application functionality on different platforms. Procedures should be implemented to isolate platform dependence features. For example, all platform-dependent procedures could be abstracted into platform-independent class library. This allows cross platform implementation of different components and enhances reasonability.

xiii. The *Scope* of the different components is well defined and user-friendliness, screen display, clarity, length of time, topic interest, and overall usability. The components should conform to well established standards.

xiv. *Development Methodology* i.e. a well-defined methodology was used throughout the system and a framework for application deployment is well defined. A consistent development methodology makes it easier to compare existing systems and documentation to uncover commonality and variability.

xv. *Security*. All security issues should be considered. While this issue has not been addressed in literature, there are many well-tested solutions that could be used. Security systems such as version control, for maintaining a new version of a file, or public-key encrypted electronic signatures may be implemented on a network system. Also, all data stored in databases must be securely protected against hardware failures such as disk crashes. The system must provide different options for providing data security and backup facilities, e.g. the use of on backup media, such as tapes or dicks.

This may help determine whether the system provides enough tools to be evaluated and used for an ITS development or to help evaluate the tools provided by the system. Existing tools are an important factor to be considered in order to foster reuse and ensure uniform development methods. Reuse potential is important to the success of any application development effort.

This determines whether there will be a future need for new or enhanced systems, and determines the need for the further development. However, component can be used for more than just the development/enhancement of an application.

APPENDIX D

EVALUATION QUESTIONNAIRE

Your Personal Details (Optional)

User Name:.....

Organisation:.....

Phone / Fax:.....

E-mail:.....

GeNisa *software aims to provide a* Generic *architecture and modular application components for Intelligent Tutoring Systems. More specifically, the GeNisa architecture aims to facilitate development and delivery of instruction by reusing existing components and is portable over different platforms. As part of the process of evaluating the GeNisa, you will have used the GeNisa software from the perspective of an ITS developer.*

The purpose of this questionnaire is to help evaluate various aspects of the GeNisa software components and environment. Your feedback will assist us in enhancing the architecture and its functionalities.

Please send completed questionnaire to:

Tajudeen Atolagbe

Department of Information Systems and Computing
Brunel University, Uxbridge.
UB8 3PH. United Kingdom.
Email: tajudeen.atolagbe@brunel.ac.uk

Instruction

Please tick all the boxes that are applicable:

1. Your Profile

1.1. What is your profession?

- ITS Researcher ☐
- Lecturer ☐
- Developer ☐
- Technician ☐
- Student ☐
- Other ☐

1.2 Which of the processes do you perform as part of your job?

- Evaluate ITS ☐
- Project Management ☐
- Software Development ☐
- Software Procurement ☐
- User Support ☐
- Other ☐

1.3 Which operating system are you using as part of your work?

- Windows ☐
- Unix ☐
- Other ☐

1.4 How often do you use an intelligent tutoring system (ITS)?

- Never ☐
- Daily ☐
- Weekly ☐
- Monthly ☐
- Less frequently than monthly ☐

1.5 How important is it that you are able to reuse ITS components and be able to deploy your application on different platforms?

- Extremely ☐
- Important ☐
- Preferred ☐
- Not Important ☐
- Irrelevant ☐

2. Suitability for Courseware Authoring and Deployment

2.1 Please indicate the extent to which you agree or disagree with each of these statements, related to GeNisa software, making use of the scale provided in each case.

	Strongly Agree	**Agreed**	**Neutral**	**Disagree**	**Strongly Disagree**
Navigation facilities are adequate	☐	☐	☐	☐	☐
Classes developed and populated independently	☐	☐	☐	☐	☐
Easy to switch between different menu levels	☐	☐	☐	☐	☐
Easy to evoke required	☐	☐	☐	☐	☐

procedures					
Easy navigation between different modules	☐	☐	☐	☐	☐
Predetermined sequence of procedures for tasks	☐	☐	☐	☐	☐
Command option for menu selection	☐	☐	☐	☐	☐
User can interrupt any dialogue at any time	☐	☐	☐	☐	☐
Commands and tools are easy to find	☐	☐	☐	☐	☐
Message boxes are contents specific	☐	☐	☐	☐	☐

2.2 Please specify how you see the implementation of the following functions within the GeNisa software and components:

	Not Good	**Moderately Good**	**Neutral**	**Good**	**Very Good**
Functions support current tasks	☐	☐	☐	☐	☐
Search engine	☐	☐	☐	☐	☐
Routines are tasks specific	☐	☐	☐	☐	☐
Clarity of the screen layout	☐	☐	☐	☐	☐
Input and output procedures	☐	☐	☐	☐	☐
Access to the knowledge base	☐	☐	☐	☐	☐
Adaptable for new tasks	☐	☐	☐	☐	☐
Screen presentation supports my work	☐	☐	☐	☐	☐
Commands are easy to find	☐	☐	☐	☐	☐
Beneficial to your work	☐	☐	☐	☐	☐
Ease of use	☐	☐	☐	☐	☐

2.3 Do you feel some key components are missing in GeNisa software.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

If you agree, please specify which facilities are missing:

_____

_____

_____

_____

_____

_____

2.4 If you have been using other ITS software similar to GeNisa, please rate the overall GeNisa functionality with relevance to these systems?

Not Good	Moderately Good	Neutral	Good	Very Good
❒	❒	❒	❒	❒

2.5 Please describe other unique functions which you have seen in these other ITS software and if appropriate list the names of these systems:

_____

_____

_____

_____

_____

_____

3.0 Unique Features and Fuctionalities

3.1 Please state your perception of the following features in GeNisa software.

	Not Good	Moderately Good	Neutral	Good	Very Good
Object oriented architecture	❒	❒	❒	❒	❒
Portable across different platforms	❒	❒	❒	❒	❒
Objects can easily be modified	❒	❒	❒	❒	❒
Exposes reusable objects to developer	❒	❒	❒	❒	❒
Reuse-oriented approach	❒	❒	❒	❒	❒
Performance on different platforms	❒	❒	❒	❒	❒
Binary interface permits runtime reuse of classes	❒	❒	❒	❒	❒
Objects promotes encapsulation and portability	❒	❒	❒	❒	❒
Provide common functionality within tools	❒	❒	❒	❒	❒
Integrate with other software package	❒	❒	❒	❒	❒
Ease of adaptation	❒	❒	❒	❒	❒
Authoring tools supports optimal usage	❒	❒	❒	❒	❒
GeNisa makes use of realistic case scenarios	❒	❒	❒	❒	❒
Ability to easily create "links" between tutorial pages	❒	❒	❒	❒	❒
Capacity to handle rich text, graphics etc	❒	❒	❒	❒	❒
Supports schema evolution	❒	❒	❒	❒	❒
Extensible	❒	❒	❒	❒	❒
GUI controls are provided to enrich development	❒	❒	❒	❒	❒
Facilitates connectivity to other databases	❒	❒	❒	❒	❒
Allows developer to add classes	❒	❒	❒	❒	❒
Includes lists of instructional strategies	❒	❒	❒	❒	❒
Scalability	❒	❒	❒	❒	❒
Flexibility	❒	❒	❒	❒	❒
Cost-effective development tool	❒	❒	❒	❒	❒
Construction is facilitated by knowledge representation	❒	❒	❒	❒	❒

Classes are domain independent	❒	❒	❒	❒	❒
Reusable for different scenarios and applicable to other domains	❒	❒	❒	❒	❒

3.2 Please rate the above features in terms of supporting your courseware development

Not Good	Moderately Good	Neutral	Good	Very Good
❒	❒	❒	❒	❒

3.3 Different courseware authoring tools are identified and represented.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.4 The components meet your courseware development needs.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.5 Easily tailor the components to your needs by assembling classes and tools rather than programming from scratch.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.6 Appropriate taxonomic characterisation of tasks that adequately supports pedagogy development.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.7 GeNisa software enables and supports reuse of previously developed knowledge bases.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.8 GeNisa software is flexible, open and provides automatic courseware generation.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

3.9 Please specify any features provided in GeNisa that are, in your opinion unique or better that those provided in other ITS software used.

_____

_____

_____

_____

_____

4.0 Module Representation

4.1 Please specify how you see the implementation of the following modules:

	Not Good	**Moderately**	**Neutral**	**Good**	**Very**

			Good			Good
Provides enough information about which entries	❒	❒	❒	❒	❒	
Contents are clear and unambiguous	❒	❒	❒	❒	❒	
Visual queues used to indicate entry point	❒	❒	❒	❒	❒	
Clarity of feedback messages	❒	❒	❒	❒	❒	
Navigation within the software	❒	❒	❒	❒	❒	
Interrupt any dialog at any time	❒	❒	❒	❒	❒	
Easy of moving between different screens	❒	❒	❒	❒	❒	
Usage of the system functionality	❒	❒	❒	❒	❒	
Abort a running procedure manually	❒	❒	❒	❒	❒	
Output of functions	❒	❒	❒	❒	❒	
Use of prompts / messages boxes	❒	❒	❒	❒	❒	

4.2 Little training is required before a user can employ GeNisa software productively.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

4.3 Interaction with the system is clear and not subject to misinterpretation.

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

4.4 Components are adequate for courseware development

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

4.5 Components have necessary functionalities

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
❒	❒	❒	❒	❒

5.0 Difficulties

5.1 Please indicate the parts of the system, which were difficult to use/understand:

Function	**Difficult to use or Understand**
Authoring tools	❒

Search engine	❐
Courseware authoring	❐
Knowledge acquisition module	❐
Editing the scenario library	❐
Retrieve information from field	❐
Messages displayed	❐
Correcting mistakes	❐
Navigation	❐
Feedback	❐

5.2 Did you find the GeNisa user interface easy or difficult to master?

Very Difficult	**Fairly Difficult**	**Neutral**	**Fairly Easy**	**Easy**
❐	❐	❐	❐	❐

5.3 Did you find it easy or difficult to navigate through the screen/windows of the GeNisa user interface?

Very Difficult	**Fairly Difficult**	**Neutral**	**Fairly Easy**	**Easy**
❐	❐	❐	❐	❐

5.4 Did you find the use of the system tiring for reasons such as: much information or not in logical order, tiring colours, difficult fonts other. If you answer Yes or Neutral please comment below:

YES	**Neutral**	**NO**
❐	❐	❐

5.5 Please rate the overall design and look of the GeNisa user interface:

Very Good	**Good**	**Neutral**	**Satisfactory**	**Poor**
❐	❐	❐	❐	❐

6.0 GeNisa Software Performance

6.1 Please rate the performance of GeNisa software in reference to the following tasks:

	Very Good	**Good**	**Neutral**	**Satisfactory**	**Poor**
Speed of response	❐	❐	❐	❐	❐
How well are text/icons/tables displayed on your screen	❐	❐	❐	❐	❐
Presentation of information on-line	❐	❐	❐	❐	❐
Contents sensitivity	❐	❐	❐	❐	❐

Use of Scenarios	☐	☐	☐	☐	☐
Courseware Authoring	☐	☐	☐	☐	☐
Portability to other platform	☐	☐	☐	☐	☐
Reuse of components and functions	☐	☐	☐	☐	☐
Designations are used consistently	☐	☐	☐	☐	☐
Meaningful message boxes	☐	☐	☐	☐	☐

6.2 Please make any specific comments or provide examples in reference to any problems you may have faced in using the GeNisa software (eg. specific problems you may have faced, etc.):

_____

_____

_____

_____

_____

6.3 The following components are adequately provided in the GeNisa:

	Strongly Agree	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Quality of Instruction	☐	☐	☐	☐	☐
User Interface	☐	☐	☐	☐	☐
Information presentation	☐	☐	☐	☐	☐
Information Integrity	☐	☐	☐	☐	☐
Dialogue/Message Boxes	☐	☐	☐	☐	☐
Knowledge Base	☐	☐	☐	☐	☐
Scenario Library	☐	☐	☐	☐	☐
On-line help	☐	☐	☐	☐	☐
Inference Engine	☐	☐	☐	☐	☐
Tutorial Discourse	☐	☐	☐	☐	☐

6.4 Please rate the level of error tolerance in the component as presented in GeNisa:

	Strongly Agree	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Small mistakes have sometimes had serious consequences	☐	☐	☐	☐	☐
Prompts are used to confirm destructive operation (e.g. deletion of file etc.),	☐	☐	☐	☐	☐
Mistakes can easily be corrected	☐	☐	☐	☐	☐
Needs minimum technical support	☐	☐	☐	☐	☐
Data are checked for correctness before processing is initiated	☐	☐	☐	☐	☐
Easily undo the last operation	☐	☐	☐	☐	☐
Error messages are helpful	☐	☐	☐	☐	☐
Warning about potential problem situations	☐	☐	☐	☐	☐
Keep the original data even after it has been changed	☐	☐	☐	☐	☐
Provides useful information on how to recover from mistakes	☐	☐	☐	☐	☐

7.0 Reuse, Portability and Learning

7.1 The following statements test the suitability for reuse, portability and learning.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Easy to adapt forms, screens and menus	☐	☐	☐	☐	☐
Faster access to components	☐	☐	☐	☐	☐
Components integration	☐	☐	☐	☐	☐
Access to data from a number of resources	☐	☐	☐	☐	☐
Greater productivity for developer on different platforms	☐	☐	☐	☐	☐
Reduction in development time	☐	☐	☐	☐	☐
Ability to access information during instruction	☐	☐	☐	☐	☐
Support for other software	☐	☐	☐	☐	☐
Reuse of data and attributes	☐	☐	☐	☐	☐
Mixed-initiative knowledge acquisition	☐	☐	☐	☐	☐
Incorporates set of tools to optimise ITS development	☐	☐	☐	☐	☐
Adjust the amount of information (data, text, graphics, etc.) displayed	☐	☐	☐	☐	☐

7.2 The following statement tests the suitability for learning

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Long time needed to learn how to use the software	☐	☐	☐	☐	☐
The explanation provided is clear and easy to understand	☐	☐	☐	☐	☐
GeNisa is intuitive	☐	☐	☐	☐	☐
Requires remembering many details	☐	☐	☐	☐	☐
The program performs better than other ITS that I have used	☐	☐	☐	☐	☐
Sufficient number of examples are given for each topic	☐	☐	☐	☐	☐
Help information in the GeNisa is clear and precise	☐	☐	☐	☐	☐
Courseware materials are comprehensive	☐	☐	☐	☐	☐
Tutorial directions are clear	☐	☐	☐	☐	☐
I have control during instruction	☐	☐	☐	☐	☐
Easily adapted to suit my own level of knowledge and domain	☐	☐	☐	☐	☐
Course is divided into smaller modules	☐	☐	☐	☐	☐
Scenarios are used appropriately	☐	☐	☐	☐	☐
Help improve knowledge of the subject	☐	☐	☐	☐	☐
Interactivity provided is adequate	☐	☐	☐	☐	☐
Meaningful error messages are provided	☐	☐	☐	☐	☐
Instruction is consistently designed, thus making it easy to use	☐	☐	☐	☐	☐

8.1 In what ways do you feel that the GeNisa Software can be improved and new facilities added?

8.2 Please use space below to make any comments or suggestions concerning the package, which may be useful for current or for further development

Thank you for completing this questionnaire.

Completed questionnaire should be returned either by post, fax or email to the address given on page 1.

GeNisa EVALUATION QUESTIONNAIRE

Your Personal Details (optional)

User Name: .....

Organisation: .....

Phone / Fax: .....

E-mail: .....

GeNisa

GeNisa software aims to provide a *Generic* architecture and modular application components for Intelligent Tutoring Systems. More specifically, the GeNisa architecture aims to facilitate development and delivery of instruction by reusing existing components and is portable over different platforms. As part of the process of evaluating the GeNisa, you will have used the GeNisa software from the perspective of an ITS developer.

The purpose of this questionnaire is to help evaluate various aspects of the GeNisa software components and environment. Your feedback will assist us in enhancing the architecture and its functionalities.

Tajudeen Atolagbe

Department of Information Systems and Computing
Brunel University, Uxbridge.
UB8 3PH. United Kingdom.
email: Tajudeen.Atolagbe@brunel.ac.uk

Instruction

Please tick all the boxes that are applicable:

1. Your Profile

1.1. What is your profession?

- ITS Researcher ☐
- Lecturer ☐
- Developer ☑
- Technician ☐
- Student ☐
- Other ☐

1.2 Which of the processes do you perform as part of your job?

- Evaluate ITS ☐
- Project Management ☐
- Software Development ☑
- Software Procurement ☐
- User Support ☐
- Other ☐

1.3 Which operating system are you using as part of your work?

- Windows ☑
- Unix ☐
- Other ☐

1.4 How often do you use an intelligent tutoring system (ITS)?

- Never ☐
- Daily ☑
- Weekly ☐
- Monthly ☐
- Less frequently than monthly ☐

1.5 How important is it that you are able to reuse ITS components and be able to deploy yo application on different platforms?

- Extremely ☑
- Important ☐
- Preferred ☐
- Not Important ☐
- Irrelevant ☐

The Suitability for Courseware Authoring and Deployment

Please indicate the extent to which you agree or disagree with each of these statements, related to GeNisa software, making use of the scale provided in each case.

	Strongly Agree	Agreed	Neutral	Disagree	Strongly Disagree
Navigation facilities are adequate	☑	☐	☐	☐	☐
Classes developed and populated independently	☑	☐	☐	☐	☐
Easy to switch between different menu levels	☑	☐	☐	☐	☐
Easy to evoke required procedures	☑	☐	☐	☐	☐
Easy navigation between different modules	☑	☐	☐	☐	☐
Predetermined sequence of procedures for tasks	☑	☐	☐	☐	☐
Command option for menu selection	☐	☑	☐	☐	☐
User can interrupt any dialogue at any time	☑	☐	☐	☐	☐
Commands and tools are easy to find	☑	☐	☐	☐	☐
Message boxes are contents specific	☑	☐	☐	☐	☐

Please specify how you see the implementation of the following functions within the GeNisa software and components:

	Not Good	Moderately Good	Neutral	Good	Very Good
Functions support current tasks	☐	☐	☐	☐	☑
Search engine	☐	☐	☐	☐	☑
Routines are tasks specific	☐	☐	☐	☐	☑
Clarity of the screen layout	☐	☐	☐	☐	☑
Input and output procedures	☐	☐	☐	☐	☑
Access to the knowledge base	☐	☐	☐	☐	☑
Adaptable for new tasks	☐	☐	☐	☐	☑

Screen presentation supports my work	☐	☐	☐	☐	☑
Commands are easy to find	☐	☐	☐	☐	☑
Beneficial to your work	☐	☐	☐	☐	☑
Ease of use	☐	☐	☐	☐	☑

2.3 Do you feel some key components are missing in GeNisa software

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☑

If you agree, please specify which facilities are missing:

2.4 If you have been using other ITS software similar to GeNisa, please rate the overall GeNisa functionality with relevance to these systems?

Not Good	Moderately Good	Neutral	Good	Very Good
☐	☐	☐	☐	☑

2.5 Please describe other unique functions which you have seen in these other ITS software and if appropriate list the names of these systems:

Unique Features and Fuctionalities

Please state your perception of the following features in GeNisa software.

	Not Good	Moderately Good	Neutral	Good	Very Good
Object oriented architecture	☐	☐	☐	☐	☑
Portable across different platforms	☐	☐	☐	☐	☑
Objects can easily be modified	☐	☐	☐	☐	☑
Exposes reusable objects to developer	☐	☐	☐	☐	☑
Reuse-oriented approach	☐	☐	☐	☐	☑
Performance on different platforms	☐	☐	☐	☑	☐
Binary interface permits runtime reuse of classes	☐	☐	☐	☐	☑
Objects promotes encapsulation and portability	☐	☐	☐	☐	☑
Provide common functionality within tools	☐	☐	☐	☐	☑
Integrate with other software package	☐	☐	☐	☐	☑
Ease of adaptation	☐	☐	☐	☑	☐
Authoring tools supports optimal usage	☐	☐	☐	☐	☑
GeNisa makes use of realistic case scenarios	☐	☐	☐	☐	☑
Ability to easily create "links" between tutorial pages	☐	☐	☑	☐	☐
Capacity to handle rich text, graphics etc	☐	☐	☐	☑	☐
Supports schema evolution	☐	☐	☐	☑	☐
Extensible	☐	☐	☐	☐	☑
GUI controls are provided to enrich development	☐	☐	☐	☐	☑
Facilitates connectivity to other databases	☐	☐	☑	☐	☐
Allows developer to add classes	☐	☐	☐	☐	☑
Includes lists of instructional strategies	☐	☐	☐	☑	☐
Scalability	☐	☐	☐	☐	☑
Flexibility	☐	☐	☐	☐	☑
Cost-effective development tool	☐	☐	☐	☐	☑
Construction is facilitated by knowledge representation	☐	☐	☐	☑	☐
Classes are domain independent	☐	☐	☐	☐	☑
Reusable for different scenarios	☐	☐	☐	☐	☑

and applicable to other domains					

3.2 Please rate the above features in terms of supporting your courseware development

Not Good	Moderately Good	Neutral	Good	Very Good
☐	☐	☐	☐	☑

3.3 Different courseware authoring tools are identified and represented

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.4 The components meet your courseware development needs

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.5 Easily tailor the components to your needs by assembling classes and tools rather than programming from scratch

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.6 Appropriate taxonomic characterisation of tasks that adequately supports pedagogy development

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.7 GeNisa software enables and supports reuse of previously developed knowledge bases

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.8 GeNisa software is flexible, open and provides automatic courseware generation

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.9 **Please specify any features provided in GeNisa that are, in your opinion unique or better than those provided in other ITS software used.**

The placement and functions of elements within the Windows are appropriate and consistent

4.0 **Module Representation**

4.1 **Please specify how you see the implementation of the following modules:**

	Not Good	Moderately Good	Neutral	Good	Very Good
Provides enough information about which entries	☐	☐	☐	☐	☑
Contents are clear and unambiguous	☐	☐	☐	☐	☑
Visual queues used to indicate entry point	☐	☐	☐	☐	☑
Clarity of feedback messages	☐	☐	☐	☐	☑
Navigation within the software	☐	☐	☐	☐	☑
Interrupt any dialog at any time	☐	☐	☐	☐	☑
Easy of moving between different screens	☐	☐	☐	☐	☑
Usage of the system functionality	☐	☐	☐	☐	☑
Abort a running procedure manually	☐	☐	☐	☐	☑
Output of functions	☐	☐	☐	☑	☐
Use of prompts / messages boxes	☐	☐	☐	☐	☑

4.2 Little training is required before a user can employ GeNisa software productively

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

4.3 Interaction with the system is clear and not subject to misinterpretation

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

4.4 Components are adequate for courseware development

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

4.5 Components have necessary functionalities

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

5.0 Difficulties

5.1 Please indicate the parts of the system which were difficult to use/understand:

Function	Difficult to use or Understand
Authoring tools	☐
Inference engine	☐
Courseware authoring	☐
Knowledge acquisition module	☐
Editing the scenario library	☐
Retrieve information from field	☐
Messages displayed	☐
Correcting mistakes	☐
Navigation	☐
Feedback	☐

5.2 **Did you find the GeNisa user interface easy or difficult to master?**

Very Difficult	Fairly Difficult	Neutral	Fairly Easy	Easy
☐	☐	☐	☐	☐

5.3 **Did you find it easy or difficult to navigate through the screen/windows of the GeNisa user interface?**

Very Difficult	Fairly Difficult	Neutral	Fairly Easy	Easy
☐	☐	☐	☐	☐

5.4 **Did you find the use of the system tiring for reasons such as: much information or not in logical order, tiring colours, difficult fonts other. If you answer Yes or Neutral please comment below:**

YES	Neutral	NO
☐	☐	☐

5.5 **Please rate the overall design and look of the GeNisa user interface:**

Very Good	Good	Neutral	Satisfactory	Poor
☐	☐	☐	☐	☐

6.0 GeNisa Software Performance

6.1 **Please rate the performance of GeNisa software in reference to the following tasks:**

	Very Good	Good	Neutral	Satisfactory	Poor
Speed of response	☐	☐	☐	☐	☐
How well are text/icons/tables displayed	☐	☐	☐	☐	☐

on your screen					
Presentation of information on-line	☑	☐	☐	☐	☐
Contents sensitivity	☑	☐	☐	☐	☐
Use of Scenarios	☑	☐	☐	☐	☐
Courseware Authoring	☐	☑	☐	☐	☐
Portability to other platform	☑	☐	☐	☐	☐
Reuse of components and functions	☑	☐	☐	☐	☐
Designations are used consistently	☐	☑	☐	☐	☐
Meaningful message boxes	☑	☐	☐	☐	☐

6.2 Please make any specific comments or provide examples in reference to any problems you may have faced in using the GeNisa software (eg. specific problems you may have faced, etc.):

None

6.3 The following components are adequately provided in the GeNisa:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Quality of Instruction	☐	☑	☐	☐	☐
User Interface	☑	☐	☐	☐	☐
Information presentation	☑	☐	☐	☐	☐
Information Integrity	☐	☑	☐	☐	☐
Dialogue/Message Boxes	☑	☐	☐	☐	☐
Knowledge Base	☑	☐	☐	☐	☐
Scenario Library	☑	☐	☐	☐	☐
On-line help	☐	☑	☐	☐	☐
Inference Engine	☑	☐	☐	☐	☐
Tutorial Discourse	☑	☐	☐	☐	☐

6.4 **Please rate the level of error tolerance in the component as presented in GeNisa:**

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Small mistakes have sometimes had serious consequences	☐	☐	☐	☐	☐
Prompts are used to confirm destructive operation (e.g. deletion of file etc.),	☐	☐	☐	☐	☐
Mistakes can easily be corrected	☐	☐	☐	☐	☐
Needs minimum technical support	☐	☐	☐	☐	☐
Data are checked for correctness before processing is initiated	☐	☐	☐	☐	☐
Easily undo the last operation	☐	☐	☐	☐	☐
Error messages are helpful	☐	☐	☐	☐	☐
Warning about potential problem situations	☐	☐	☐	☐	☐
Keep the original data even after it has been changed	☐	☐	☐	☐	☐
Provides useful information on how to recover from mistakes	☐	☐	☐	☐	☐

7.0 **Reuse, Portability and Learning**

7.1 **The following statements test the suitability for reuse, portability and learning**

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Easy to adapt forms, screens and menus	☐	☐	☐	☐	☐
Faster access to components	☐	☐	☐	☐	☐
Components integration	☐	☐	☐	☐	☐
Access to data from a number of resources	☐	☐	☐	☐	☐
Greater productivity for developer on different platforms	☐	☐	☐	☐	☐
Reduction in development time	☐	☐	☐	☐	☐
Ability to access information during instruction	☐	☐	☐	☐	☐
Support for other software	☐	☐	☐	☐	☐
Reuse of data and attributes	☐	☐	☐	☐	☐
Mixed-initiative knowledge acquisition	☐	☐	☐	☐	☐
Incorporates set of tools to optimise	☐	☐	☐	☐	☐

ITS development					
Adjust the amount of information (data, text, graphics, etc.) displayed	❒	❒	❒	❒	❒

7.2 The following statement tests the suitability for learning

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Long time needed to learn how to use the software	❒	❒	❒	❒	❒
The explanation provided is clear and easy to understand	❒	❒	❒	❒	❒
GeNisa is intuitive	❒	❒	❒	❒	❒
Requires remembering many details	❒	❒	❒	❒	❒
The program performs better than other ITS that I have used	❒	❒	❒	❒	❒
Sufficient number of examples are given for each topic	❒	❒	❒	❒	❒
Help information in the GeNisa is clear and precise	❒	❒	❒	❒	❒
Courseware materials are comprehensive	❒	❒	❒	❒	❒
Tutorial directions are clear	❒	❒	❒	❒	❒
I have control during instruction	❒	❒	❒	❒	❒
Easily adapted to suit my own level of knowledge and domain	❒	❒	❒	❒	❒
Course is divided into smaller modules	❒	❒	❒	❒	❒
Scenarios are used appropriately	❒	❒	❒	❒	❒
Help improve knowledge of the subject	❒	❒	❒	❒	❒
Interactivity provided is adequate	❒	❒	❒	❒	❒
Meaningful error messages are provided	❒	❒	❒	❒	❒
Instruction is consistently designed, thus making it easy to use	❒	❒	❒	❒	❒

1 **In what ways do you feel that the GeNisa Software can be improved and new facilities added?**

Could provide animated characters to support teaching

2 **Please use space below to make any comments or suggestions concerning the package, which may be useful for current or for further development**

Integration with other productivity tools e.g. spreadsheet, database, PowerPoint

Thank you for completing this questionnaire.

Completed questionnaire should be returned either by post, fax or email to the address given on page 1.

ITS development					
Adjust the amount of information (data, text, graphics, etc.) displayed	☐	☐	☐	☐	☐

7.2 The following statement tests the suitability for learning

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Long time needed to learn how to use the software	☐	☐	☐	☐	☐
The explanation provided is clear and easy to understand	☐	☐	☐	☐	☐
GeNisa is intuitive	☐	☐	☐	☐	☐
Requires remembering many details	☐	☐	☐	☐	☐
The program performs better than other ITS that I have used	☐	☐	☐	☐	☐
Sufficient number of examples are given for each topic	☐	☐	☐	☐	☐
Help information in the GeNisa is clear and precise	☐	☐	☐	☐	☐
Courseware materials are comprehensive	☐	☐	☐	☐	☐
Tutorial directions are clear	☐	☐	☐	☐	☐
I have control during instruction	☐	☐	☐	☐	☐
Easily adapted to suit my own level of knowledge and domain	☐	☐	☐	☐	☐
Course is divided into smaller modules	☐	☐	☐	☐	☐
Scenarios are used appropriately	☐	☐	☐	☐	☐
Help improve knowledge of the subject	☐	☐	☐	☐	☐
Interactivity provided is adequate	☐	☐	☐	☐	☐
Meaningful error messages are provided	☐	☐	☐	☐	☐
Instruction is consistently designed, thus making it easy to use	☐	☐	☐	☐	☐

11 **In what ways do you feel that the GeNisa Software can be improved and new facilities added?**

Could provide connected services to support teaching

12 **Please use space below to make any comments or suggestions concerning the package, which may be useful for current or for further development**

Integration with other productivity tools e.g. spreadsheet, statistic packages

Thank you for completing this questionnaire.

Completed questionnaire should be returned either by post, fax or email to the address given on page 1.

GeNisa EVALUATION QUESTIONNAIRE

Your Personal Details (optional)

User Name: .....

Organisation: .....

Phone / Fax: .....

E-mail: .....

GeNisa

GeNisa software aims to provide a *Generic* architecture and modular application components for Intelligent Tutoring Systems. More specifically, the GeNisa architecture aims to facilitate development and delivery of instruction by reusing existing components and is portable over different platforms. As part of the process of evaluating the GeNisa, you will have used the GeNisa software from the perspective of an ITS developer.

The purpose of this questionnaire is to help evaluate various aspects of the GeNisa software components and environment. Your feedback will assist us in enhancing the architecture and its functionalities.

Tajudeen Atolagbe

Department of Information Systems and Computing
Brunel University, Uxbridge.
UB8 3PH. United Kingdom.
email: Tajudeen.Atolagbe@brunel.ac.uk

Instruction

Please tick all the boxes that are applicable:

1. Your Profile

1.1. What is your profession?

- ITS Researcher ☐
- Lecturer ☐
- Developer ☐
- Technician ☐
- Student ☐
- Other ☑

1.2 Which of the processes do you perform as part of your job?

- Evaluate ITS ☐
- Project Management ☐
- Software Development ☑
- Software Procurement ☐
- User Support ☐
- Other ☐

1.3 Which operating system are you using as part of your work?

- Windows ☑
- Unix ☐
- Other ☑

1.4 How often do you use an intelligent tutoring system (ITS)?

- Never ☐
- Daily ☐
- Weekly ☐
- Monthly ☑
- Less frequently than monthly ☐

1.5 How important is it that you are able to reuse ITS components and be able to deploy your application on different platforms?

- Extremely ☐
- Important ☑
- Preferred ☐
- Not Important ☐
- Irrelevant ☐

The Suitability for Courseware Authoring and Deployment

Please indicate the extent to which you agree or disagree with each of these statements, related to GeNisa software, making use of the scale provided in each case.

	Strongly Agree	Agreed	Neutral	Disagree	Strongly Disagree
Navigation facilities are adequate	☑	☐	☐	☐	☐
Classes developed and populated independently	☑	☐	☐	☐	☐
Easy to switch between different menu levels	☑	☐	☐	☐	☐
Easy to evoke required procedures	☐	☑	☐	☐	☐
Easy navigation between different modules	☑	☐	☐	☐	☐
Predetermined sequence of procedures for tasks	☑	☐	☐	☐	☐
Command option for menu selection	☑	☐	☐	☐	☐
User can interrupt any dialogue at any time	☑	☐	☐	☐	☐
Commands and tools are easy to find	☐	☑	☐	☐	☐
Message boxes are contents specific	☐	☐	☑	☐	☐

Please specify how you see the implementation of the following functions within the GeNisa software and components:

	Not Good	Moderately Good	Neutral	Good	Very Good
Functions support current tasks	☐	☐	☐	☑	☐
Search engine	☐	☐	☐	☐	☑
Routines are tasks specific	☐	☐	☐	☐	☑
Clarity of the screen layout	☐	☐	☐	☐	☑
Input and output procedures	☐	☐	☐	☐	☑
Access to the knowledge base	☐	☐	☐	☑	☐
Adaptable for new tasks	☐	☐	☐	☐	☑

Screen presentation supports my work	☐	☐	☐	☐	☑
Commands are easy to find	☐	☐	☐	☐	☑
Beneficial to your work	☐	☐	☐	☐	☑
Ease of use	☐	☐	☐	☐	☑

2.3 Do you feel some key components are missing in GeNisa software

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☑	☐

If you agree, please specify which facilities are missing:

2.4 If you have been using other ITS software similar to GeNisa, please rate the overall GeNisa functionality with relevance to these systems?

Not Good	Moderately Good	Neutral	Good	Very Good
☐	☐	☐	☐	☑

2.5 Please describe other unique functions which you have seen in these other ITS software and if appropriate list the names of these systems:

good screen location
~~strategic~~ programme

3.0 Unique Features and Fuctionalities

3.1 Please state your perception of the following features in GeNisa software.

	Not Good	Moderately Good	Neutral	Good	Very Good
Object oriented architecture	☐	☐	☐	☐	☑
Portable across different platforms	☐	☐	☐	☐	☑
Objects can easily be modified	☐	☐	☐	☐	☑
Exposes reusable objects to developer	☐	☐	☐	☑	☐
Reuse-oriented approach	☐	☐	☐	☐	☑
Performance on different platforms	☐	☐	☐	☐	☑
Binary interface permits runtime reuse of classes	☐	☐	☑	☐	☐
Objects promotes encapsulation and portability	☐	☐	☐	☐	☑
Provide common functionality within tools	☐	☐	☐	☐	☑
Integrate with other software package	☐	☐	☐	☐	☑
Ease of adaptation	☐	☐	☐	☐	☑
Authoring tools supports optimal usage	☐	☐	☐	☐	☑
GeNisa makes use of realistic case scenarios	☐	☐	☐	☐	☑
Ability to easily create "links" between tutorial pages	☐	☐	☐	☑	☐
Capacity to handle rich text, graphics etc	☐	☐	☐	☑	☐
Supports schema evolution	☐	☐	☑	☐	☐
Extensible	☐	☐	☐	☐	☑
GUI controls are provided to enrich development	☐	☐	☐	☐	☑
Facilitates connectivity to other databases	☐	☐	☐	☐	☑
Allows developer to add classes	☐	☐	☐	☐	☑
Includes lists of instructional strategies	☐	☐	☐	☑	☐
Scalability	☐	☐	☐	☐	☑
Flexibility	☐	☐	☐	☑	☐
Cost-effective development tool	☐	☐	☐	☐	☑
Construction is facilitated by knowledge representation	☐	☐	☐	☑	☐
Classes are domain independent	☐	☐	☐	☐	☑
Reusable for different scenarios	☐	☐	☐	☐	☑

and applicable to other domains					

3.2 Please rate the above features in terms of supporting your courseware development

Not Good	Moderately Good	Neutral	Good	Very Good
☐	☐	☐	☐	☑

3.3 Different courseware authoring tools are identified and represented

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.4 The components meet your courseware development needs

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.5 Easily tailor the components to your needs by assembling classes and tools rather than programming from scratch

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.6 Appropriate taxonomic characterisation of tasks that adequately supports pedagogy development

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

3.7 GeNisa software enables and supports reuse of previously developed knowledge bases

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.8 GeNisa software is flexible, open and provides automatic courseware generation

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☑	☐	☐	☐

3.9 Please specify any features provided in GeNisa that are, in your opinion unique or better that those provided in other ITS software used.

JAVA DevelopMent environment
To Do List / management.

4.0 Module Representation

4.1 Please specify how you see the implementation of the following modules:

	Not Good	Moderately Good	Neutral	Good	Very Good
Provides enough information about which entries	☐	☐	☐	☐	☑
Contents are clear and unambiguous	☐	☐	☐	☐	☑
Visual queues used to indicate entry point	☐	☐	☐	☐	☑
Clarity of feedback messages	☐	☐	☐	☑	☐
Navigation within the software	☐	☐	☐	☐	☑
Interrupt any dialog at any time	☐	☐	☐	☐	☑
Easy of moving between different screens	☐	☐	☐	☑	☐
Usage of the system functionality	☐	☐	☐	☐	☑
Abort a running procedure manually	☐	☐	☐	☐	☑
Output of functions	☐	☐	☐	☐	☑
Use of prompts / messages boxes	☐	☐	☐	☐	☑

4.2 **Little training is required before a user can employ GeNisa software productively**

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

4.3 **Interaction with the system is clear and not subject to misinterpretation**

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

4.4 **Components are adequate for courseware development**

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☑	☐	☐	☐	☐

4.5 **Components have necessary functionalities**

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
☐	☐	☐	☐	☐

5.0 Difficulties

5.1 Please indicate the parts of the system which were difficult to use/understand:

Function	Difficult to use or Understand
Authoring tools	☑
Inference engine	☐
Courseware authoring	☐
Knowledge acquisition module	☑
Editing the scenario library	☐
Retrieve information from field	☐
Messages displayed	☐
Correcting mistakes	☐
Navigation	☐
Feedback	☐

5.2 Did you find the GeNisa user interface easy or difficult to master?

Very Difficult	Fairly Difficult	Neutral	Fairly Easy	Easy
☐	☐	☐	☑	☐

5.3 Did you find it easy or difficult to navigate through the screen/windows of the GeNisa user interface?

Very Difficult	Fairly Difficult	Neutral	Fairly Easy	Easy
☐	☐	☐	☑	☐

5.4 Did you find the use of the system tiring for reasons such as: much information or not in logical order, tiring colours, difficult fonts other. If you answer Yes or Neutral please comment below:

YES	Neutral	NO
☐	☐	☑

_____

_____

_____

_____

_____

_____

5.5 Please rate the overall design and look of the GeNisa user interface:

Very Good	Good	Neutral	Satisfactory	Poor
☐	☑	☐	☐	☐

6.0 GeNisa Software Performance

6.1 Please rate the performance of GeNisa software in reference to the following tasks:

	Very Good	Good	Neutral	Satisfactory	Poor
Speed of response	☑	☐	☐	☐	☐
How well are text/icons/tables displayed	☑	☐	☐	☐	☐

on your screen					
Presentation of information on-line	☑	☐	☐	☐	☐
Contents sensitivity	☑	☐	☐	☐	☐
Use of Scenarios	☐	☑	☐	☐	☐
Courseware Authoring	☐	☐	☐	☐	☐
Portability to other platform	☑	☐	☐	☐	☐
Reuse of components and functions	☐	☑	☐	☐	☐
Designations are used consistently	☑	☐	☐	☐	☐
Meaningful message boxes	☑	☐	☐	☐	☐

6.2 **Please make any specific comments or provide examples in reference to any problems you may have faced in using the GeNisa software (eg. specific problems you may have faced, etc.):**

None.

6.3 **The following components are adequately provided in the GeNisa:**

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Quality of Instruction	☑	☐	☐	☐	☐
User Interface	☑	☐	☐	☐	☐
Information presentation	☐	☑	☐	☐	☐
Information Integrity	☑	☐	☐	☐	☐
Dialogue/Message Boxes	☑	☐	☐	☐	☐
Knowledge Base	☑	☐	☐	☐	☐
Scenario Library	☐	☑	☐	☐	☐
On-line help	☑	☐	☐	☐	☐
Inference Engine	☑	☐	☐	☐	☐
Tutorial Discourse	☑	☐	☐	☐	☐

ITS development					
Adjust the amount of information (data, text, graphics, etc.) displayed	☑	☐	☐	☐	☐

12 **The following statement tests the suitability for learning**

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Long time needed to learn how to use the software	☐	☐	☐	☐	☑
The explanation provided is clear and easy to understand	☑	☐	☐	☐	☐
GeNisa is intuitive	☑	☐	☐	☐	☐
Requires remembering many details	☑	☐	☐	☐	☐
The program performs better than other ITS that I have used	☑	☐	☐	☐	☐
Sufficient number of examples are given for each topic	☐	☐	☑	☐	☐
Help information in the GeNisa is clear and precise	☐	☑	☐	☐	☐
Courseware materials are comprehensive	☐	☐	☑	☐	☐
Tutorial directions are clear	☐	☐	☐	☐	☐
I have control during instruction	☑	☐	☐	☐	☐
Easily adapted to suit my own level of knowledge and domain	☑	☐	☐	☐	☐
Course is divided into smaller modules	☑	☐	☐	☐	☐
Scenarios are used appropriately	☐	☑	☐	☐	☐
Help improve knowledge of the subject	☑	☐	☐	☐	☐
Interactivity provided is adequate	☑	☐	☐	☐	☐
Meaningful error messages are provided	☐	☑	☐	☐	☐
Instruction is consistently designed, thus making it easy to use	☐	☑	☐	☐	☐

APPENDIX E

EVALUATION DATA

E.1 Introduction

This section contains analysis of data obtained from the evaluation questionnaire.

E.2 Cronbach's Coefficient Alpha

Cronbach's Coefficient Alpha (Cortina, 1993) was calculated by using the formula:

Cronbach's Coefficient Alpha: $\frac{N}{N-1} x \left(1 - \frac{\sum \left(s_x^2 \right)}{s_t^2} \right)$

Where $\left(s_x^2 \right)$ is the variance of item i. $\left(s_t^2 \right)$ is the variance of the total scores.

N is the number of items or samples.

Table E.1 Cronbach's Coefficient Alpha

Evaluation Criteria	**Weighted Score**	**Mean (X)**	**Variance (X-M)**	**$(X-M)^2$**
Suitability for courseware authoring/development	50	40.44	-11.18	124.89
Components implementation methods	65	48.56	-3.06	9.36
Unique features and functionality	170	142.89	91.27	8330.21
Module representation	86	66.78	15.16	229.82
Difficulties encountered	22	17.00	-34.62	1198.54
Overall design and look of the user interface	5	4.33	-47.28	2236.34
GeNisa performance	50	43.33	-8.29	68.72
Components representation	50	41.89	-9.73	94.67
Error tolerance	50	41.67	-9.95	99.00
Reuse and portability	60	51.44	-0.17	0.0324
Learning environment	85	69.48	17.86	318.80
Total	671	567.81	0.01	12710.38
Mean (M)	63	51.62		
Variance				246.23

Variance: $V(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{m})^2$

From the above table, $\frac{N}{N-1}x\left(1-\frac{\sum\left(s_x^2\right)}{s_t^2}\right) = 0.99.$

Kline (1993) notes that Cronbach's Coefficient Alpha should never be below 0.7 for internal consistency estimate. This is important because it takes into account variance attributable to subjects and items between them (Cortina, 1993).

E.3 Pearson's Product-Moment Correlation Coefficient

The Pearson Product-Moment Correlation Coefficient was used to find the degree that two variables "go together".

The Pearson's correlation coefficient is a number that summarises the direction and degree (closeness) of linear relations between two variables (Coolican, 1990). The sample value is represented by *r*, and the population value is represented by r (rho). The correlation coefficient can take values between -1 through 0 to +1. The sign (+ or -) of the correlation affects its interpretation. When the correlation is positive ($r > 0$), as the value of one variable increases, so does the other (Coolican, 1990).

Pearson's Product-Moment Correlation Coefficient (Coolican, 1990) was calculated by using the formal:

$$\text{Pearson's } r = \frac{Nx\sum(XxY) - \sum X x\sum Y}{\sqrt{\left[Nx\sum X^2 - (\sum X)^2\right]\left[Nx\sum Y^2 - (\sum Y)^2\right]}}$$

Where X, stands for standard scores, Y stands for the score of the second variable and N stands for number if items.

Table E.2: Pearson's Correlation Coefficient

Evaluation Criteria	Weighted (X)	Scores (Y)	(XY)	X^2	Y^2
Suitability for courseware authoring/development	50	40.44	2022	2500	1635.39
Components implementation	65	48.56	31564	4225	2358.07

methods					
Unique features and functionality	170	142.89	24291.3	28900	20417.55
Module representation	86	66.78	5743.08	7396	4459.56
Difficulties encountered	22	17.00	374	484	289
Overall design and look of the user interface	5	4.33	21.65	25	18.75
GeNisa performance	50	43.33	2166.6	2500	1877.49
Components representation	50	41.89	2094.5	2500	1754.78
Error tolerance	50	41.67	2083.5	2500	1736.39
Reuse and portability	60	51.44	3086.4	3600	2646.07
Learning environment	85	69.48	5902.4	7225	4821.91
Total (Σ)	$\Sigma X = 671$	$\Sigma Y = 567.77$	$\Sigma XY =$ 79349.33	$\Sigma X^2 = 61855$	$\Sigma Y^2 =$ 42014.05

Using Pearson's equation and the data from the table,

$$r = \frac{Nx\sum(XxY) - \sum X x \sum Y}{\sqrt{\left[Nx\sum X^2 - (\sum X)^2\right]\left[Nx\sum Y^2 - (\sum Y)^2\right]}} = 0.919$$

APPENDIX F

HEURISTIC EVALUATION

COMMENTS AND OBSERVATIONS LOG SHEET

Reviewer:..... Date:..... Log Sheet No.....

Introduction: Work through the GeNisa software. Read the systems overview. Manipulate the components and finally comment on each of the items in the table. The table is based on guideline for heuristic evaluation (Nielsen, 1994).

Table F.1 Heuristics Evaluation Tasks

Activity	Systems Area
1	The flow of information from screen to screen and on each screen evaluate.
2	Use simple and natural dialogue.
3	Speak the user's language.
4	Minimise user memory load.
5	Be consistent.
6	Provide feedback.
7	Provide Clearly Marked exits.
8	Provide shortcuts.
9	Provide good error messages.
10	Prevent errors.

Comments:

.....

.....

.....

.....

.....

.....

.....

.....

Observations:

HEURISTIC EVALUATION

COMMENTS AND OBSERVATIONS LOG SHEET

Reviewer:..... Date:..... Log Sheet No.....

Introduction: Work through the GeNisa software. Read the systems overview. Manipulate the components and finally comment on each of the items in the table. The table is based on guideline for heuristic evaluation (Nielsen, 1994).

Table F.1 Heuristics Evaluation Tasks

Activity	Systems Area
1	The flow of information from screen to screen and on each screen evaluate.
2	Use simple and natural dialogue.
3	Speak the user's language.
4	Minimise user memory load.
5	Be consistent.
6	Provide feedback.
7	Provide Clearly Marked exits.
8	Provide shortcuts.
9	Provide good error messages.
10	Prevent errors.

Comments:

Nice simplistic visual elements specifically geared towards user

Very good learner control throughout the GeNisa prototype

Component design environment is an excellent concept

Observations:

You could consider linking the strategies to the resource links for the elaborated rationale

You could add a glossary of component classes or as a element of the main window

Try to consistently distinguish between the text for the different levels of student during instruction

Have the database include information showing between several tutors for one student

HEURISTIC EVALUATION

COMMENTS AND OBSERVATIONS LOG SHEET

Reviewer..... **Date:.....** **Log Sheet No.....**

Introduction: Work through the GeNisa software. Read the systems overview. Manipulate the components and finally comment on each of the items in the table. The table is based on guideline for heuristic evaluation (Nielsen, 1994).

Table F.1 Heuristics Evaluation Tasks

Activity	Systems Area
1	The flow of information from screen to screen and on each screen evaluate.
2	Use simple and natural dialogue.
3	Speak the user's language.
4	Minimise user memory load.
5	Be consistent.
6	Provide feedback.
7	Provide Clearly Marked exits.
8	Provide shortcuts.
9	Provide good error messages.
10	Prevent errors.

Comments:

Excellent determination of screen location, menu bars etc

Very good license color throughout the ~~generate~~ GeNisa prototype

Excellent screen location throughout the programme

Observations:

Try to consistently differentiate between text for different levels of chunks during instruction

Build a way for developers workspace to be shared on local network and over the internet

APPENDIX G

INFORMAL ENQUIRIES

From:
To: <tajudeen.atolagbe@brunel.ac.uk>
Sent: October 29, 1998 12:26 PM
Subject: Re: ITS Components

Dear Taju,

We have read your abstract in ESS. Here are a few comments about the contents.

First, we are impressed by the very high quality of the paper and the objectives of the research. The component architecture as well as the resulting intelligent tutoring system component library are innovative and feasible both on the research and the development side.

We particularly liked the combined use of bayesian networks, and symbolic inference mechanisms as well as the Web and networking aspects of your paper.

. is well positioned for the project as the component architecture itself (the major part of the project) can be developed relatively easily as an extension to our Java based web agent programming infrastructure. That is by providing the networking infrastructure (Web-enabled and currently interoperating with Corba and on the way to support multicasting) as well as accelerated inference mechanisms.

Our estimate for developing the reusable intelligent component architecture, amounts to a 3-5 programmer/year effort feasible over a 1-year period. Also, tutoring system application level component of the project would require an additional effort.

We would like to support the development, release, commercialization, installation training etc., of this research. We can provide iterative, with quick prototype delivery, alpha-beta-production-maintenance milestones and quick reaction to customer
feedback and possibly evolving requirements.

We are looking forward for your feedback on this proposal, as well as on the proposed scenarios and/or designated component development options. We hope that this project can become a basis for long-standing cooperation with your company.

Best regards,

President

Message 4:
From mailer-daemon@brunel.ac.uk Tue Dec 10 23:46:53 1996
Delivery-Date: Tue, 10 Dec 1996 23:46:51 +0000
Date: Wed, 11 Dec 1996 10:17 +1030
From:
To: Tajudeen.Atolagbe@brunel.ac.uk
Subject:
X-charset: US-ASCII

Hi,

Hello, my name is : and work for the Defence Science and Technology Organisation in Australia. I am looking for applicable technologies/software for generic discrete event simulation modelling to be applied to a variety of applications eg, wide area surveillance.

I noticed the abstract of your paper for WSC 96. I would be very interested in reading your full paper. Are you able to email me a copy? Also if you are able to give me advice in the above area I would be gratefull.

Regards,