# Evolving Preferences Among Emergent Groups of Agents

Paul Marrow, Cefn Hoile, Fang Wang, and Erwin Bonsma

BTexact Technologies
Intelligent Systems Laboratory,
Adastral Park, Ipswich IP5 3RE, UK
{paul.marrow, cefn.hoile, fang.wang, erwin.bonsma}@bt.com

**Abstract.** Software agents can prove useful in representing the interests of human users of agent systems. When users have diverse interests, the question arises as to how agents representing their interests can be grouped so as to facilitate interaction between users with compatible interests. This paper describes experiments in the DIET (Decentralised Information Ecosystem Technologies) agent platform that use evolutionary computation to evolve preferences of agents in choosing environments so as to interact with other agents representing users with similar interests. These experiments suggest a useful way for agents to acquire preferences for formation of groups for information interaction between users, and may also indicate means for supporting load balancing in distributed systems.

## 1 Introduction

Software agents have proved useful for representing the interests of individual human users (e.g. [10]). With multi-agent systems there arises the possibility of managing processes on behalf of large populations of human users. Associated with this is the problem of ensuring that users with common interests get to interact appropriately. This is the issue of group formation, part of the more general problems of cooperation and collaboration in multi-agent systems.

Group formation and the associated issues of cooperation and collaboration have proved relevant to much research in multi-agent systems (e.g. [9, 11, 21]). Similar problems exist in robotics (e.g. [15]). One particular focus of research has considered how evolutionary algorithms can be used to adapt agent behaviour, and achieve collaborative or cooperative solutions ([7, 6, 16]). The use of evolutionary algorithms seems particularly appropriate in this context since they depend upon the interaction of many individuals for their success [1].

In this paper we describe how an evolutionary algorithm can be used to adapt agent behaviour in the DIET (Decentralised Information Ecosystem Technologies) system ([8, 14]), resulting in the emergence of groups of agents that share common interests. The DIET system implements large numbers of lightweight agents that can interact in a decentralised and extensible manner. The DIET system has been inspired by the interaction of organisms in natural ecosystems, and, inspired by the role of evolution in such interactions, the mechanism we use for group formation is the evolution of preferences for different environments. In the software agent context, an environment refers to the software environment an agent inhabits. In this context we assume that each environment exists on a single machine. For mobile agents, multiple environments may be on different machines. In a DIET network different environments maintain connections with each other in a "peer to peer" fashion.

It is well known that a degree of centralisation in peer to peer networks can improve the efficiency of functions such as indexing and retrieval [18]. However, existing strategies for centralisation often depend upon the existence of reliable, well known, central servers. Here we demonstrate the emergence of centralisation within a network of peers with no central servers. The dynamic approach described offers a compromise between the robustness and self-sufficiency of fully decentralised networks of transient peers with the efficiency of a centralised system.

The dynamic formation of communities of agents could be very important for the proper exploitation of computational and informational resources in future networks. The most rapid and effective interactions between agents typically are those that take place locally, between agents occupying a single environment. Accordingly we use an evolutionary algorithm to evolve preferences that lead to agents with common interests sharing the same environment.

Use of an evolutionary algorithm allows local interactions between agents to be taken advantage of in shaping the strategies used for despatch of agents to different environments over a sequence of iterative steps (evolutionary generations). Working with two populations of agents, User agents, representing user interests, and Scout agents, searching out preferred environments, we use the evolutionary algorithm to evolve the preferences of Scout agents for environments in a network of multiple environments in which agents can exist. We show that the evolutionary algorithm can increase the effectiveness of Scout agents in locating environments that are suitable for information transfer with other agents representing common interests. This can provide a basis for the automatic formation of groups of users sharing interests.

We also consider how the results from the process of group formation indicate the robustness and flexibility of the DIET system. As well as explicit selection of agents through an evolutionary algorithm, we consider how characteristics of the DIET agent environment can stimulate a process of implicit evolution, that is evolution with respect to computational resource constraints, where computational efficiency is associated with survival. This could also be used to evolve agents that adopt computationally efficient behaviour.

## 2 The DIET Platform

The experiments presented here use the DIET system ([8, 14]), a software platform that has been developed to enable the implementation of multi-agent systems consisting of very large numbers of lightweight agents, under decentralised control, interacting in a manner inspired by natural ecosystems. The development effort within the DIET project [3] is focused on providing an extensible framework for the exploration of ecologically inspired software solutions in an open agent platform.

### 2.1 Aims and Inspiration

Inspiration for the DIET system has come from natural ecosystems, where many living organisms and abiotic elements interact to produce a variety of emergent phenomena [22]. These biological systems have inspired the Universal Information Ecosystems initiative of the European Union [4], which addresses the issue of managing and understanding the complexity of the emerging global information infrastructure by looking at local and bottom-up interactions between elements of this infrastructure, in the manner of interactions between living organisms. Such local and bottom-up approaches may be expected to provide more flexibility, adaptability and scalability in response to changing circumstances than more top-down or centralised approaches. The DIET project forms part of the Universal Information Ecosystems initiative and hence the system design attempts to take these ideas into account.

### 2.2 Architecture

The DIET system is designed around a three-layer architecture [14]:

– *Core layer*: The functionality supported by the lowest layer is deliberately minimal, but designed to provide a robust and reliable service [8, 14]. It is here that the services and functions common to all agents are provided.
– *ARC layer*: Additional utilities are distributed along with the core layer, known as "Application Reusable Components" (ARC). These provide primitives that exploit the minimal functions of the core to support higher level activities common to many, but not all, applications. These include remote communication, agent reusable behaviours, multicasting and directory services.
– *Application layer*: This layer comprises additional data structures and agent behaviours for application-specific objectives.

The three-layer structure provides flexibility for implementing a variety of applications using the same core features of the agents and the software environment that they inhabit. It has been implemented in Java.

### 2.3 Core Layer

The core layer provides for Environments that are the basic units which DIET agents can inhabit. One or more Environment may be located within a DIET World, there being a single Java Virtual Machine for each World. The possibility exists for multiple Worlds in conjunction with multiple Java Virtual Machines, allowing for indefinite scaling up of the DIET system.

Each Environment provides minimal functionality to all agents, allowing for agent creation, agent destruction, local communication between agents, and initiation of migration between Environments. These basic functions have been designed so as to minimise the computational overhead required for their execution. The CPU time required for each function is not dependent upon the number of agents occupying the Environment, allowing efficient and rapid operation even in Environments inhabited by large numbers of agents. Operation of the DIET system is based upon these basic functions and the resulting local interactions between agents.

Local communication is central to local interaction between DIET agents. Local communication in this context involves the passing of messages and objects between two agents in the same Environment. The agent that initiates the communication must identify the agent that it wishes to contact - this can be done using a binary “name tag” associated with the target agent that is randomly generated in its original Environment. In addition an agent has a “family tag” that indicates the group of agents to which it belongs. These are in consequence not typically unique, but may also be used for identification. Identification of agents by either of these methods is decentralised, being associated only with particular Environments, and thus scales well with larger systems.

Once a target agent has been identified, a Connection is formed between the two agents, allowing messages and/or objects to be passed between the two agents. Each agent has a message buffer that provides a space into which messages can be received. More information about local communication is given by Hoile et al. [8]. Remote communication, that is, communication between Environments, is also possible. The core layer provides only agent migration at the Environment level. Key functions associated with remote communication are provided in the ARC layer.

## 2.4 ARC Layer

The ARC layer provides for various extensions that can support remote communication between Environments, as well as other functions. These include “Carrier Pigeon” agents that can migrate to another Environment and then deliver a message by local communication to the intended target agent. Alternatively, Mirror agents can be created in an Environment to support a communication channel to an agent in another Environment, via Carrier Pigeons that only the Mirror agent, and not the agent initiating the remote communication, interacts with. Remote communication via a Mirror agent looks like local communication to other agents in the Environment. Such means of remote communication allow for increased flexibility in interaction between agents distributed across multiple environments [8].

### 2.5 Applications

Based on the functionality provided by the core layer and the ARC layer, applications can be developed based on the DIET platform, with application-specific code situated in the third, application, layer. Examples of work in this area include [5, 12]. Applications can also take advantage of visualisation and interactive control software that is being developed [13]. The basing of application development on this architecture centred on local interactions between agents makes the DIET system particularly appropriate for the study of phenomena emerging from local interactions. We now go on to do this in the context of emerging preferences for environments supporting co-location for information sharing.

## 3 Experiments

We seek to generate emergent phenomena among agents in getting them to evolve preferences for particular environments (that are DIET Environments). As such agents can represent the interests of human users, this may be a useful mechanism for automatically ensuring that users' interests in terms of environmental preferences are best served.

We consider a situation where human users of an information management system connect transiently to a peer-to-peer network in order to identify information resources that satisfy their requirements for information. We assume that each user has a "category of interest" that represents some topic that they are particularly interested in. Users that are interested in the same category of interest are assumed to be interested in the same set of information, but to only have access to a subset of that initially. We also assume that users are interested in finding other users with the same category of interest and sharing information with them. Each user creates a DIET Environment from which agents can be created to facilitate the user's requirements.

### 3.1 World, Environments and Links

The experiments take place in the context of a DIET World composed of multiple Environments as described above (Section 2.3). Each Environment is distinct from others in terms of its distinctive signature provided by a hashcode. The Environment's hashcode is generated based on the Environment's address in the DIET World. A 32 bit hashcode is used, because a hashcode of this form can easily be acquired from all Java objects. But this form of hashcode can be replaced by one of many other hashing schemes if required (see e.g. [17]).

In our experiments Environments are connected in a peer network. This network is formed by choosing pairs of Environments at random, and then creating neighbourhood links between them. Such links are uni-directional, but link formation can be repeated to give the effect of a bi-directional link. This process is repeated until each Environment has on average four links. This level of connectivity is intended to approximate the connectivity of a fully decentralised peer network. The existence of such links between Environments allows agents to explore the network. Although agents can migrate to any Environment, they need to know the address of the destination Environment. At each Environment, agents can get the address of one of the neighbouring Environments and subsequently migrate to it and thus explore the collection of Environments. Figure 1 illustrates what such an environment network might look like.

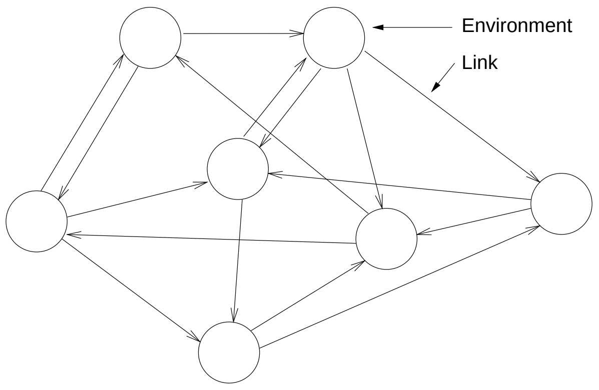

**Fig. 1.** An example DIET peer network

## 3.2 Agents

The experiments depend upon two populations of agents: User agents and Scout agents. These agents are lightweight and use the minimal functions supplied by the DIET platform. User agents represent human users and deploy and evolve Scout agents, as well as collecting information from Scout agents. Scout agents explore multiple Environments and carry out the activities needed to form groups.

Only one User agent is associated with a particular Environment. The User agent remains in that Environment throughout each experiment. Each experiment starts with a number of User agents distributed, one at each of the Environments.

Each User agent creates a population of Scout agents, and evolves them independently of other populations of Scout agents. Having created these Scout agents, the User agent dispatches them into the peer network of Environments, where they may interact with other Scout agents and other User agents, before returning to interact with the User agent that created them.

## 3.3 Evolutionary Algorithm

Scout agents are bred by User agents. User agents seek to maximise Scout agents' success in locating other Scout agents with common interests. A Scout agent's preference for Environments is determined by a bitstring genome provided at birth. A Steady State genetic algorithm is used [19], implemented using the Eos evolutionary and ecosystem platform [2]. Tournament selection, two-point crossover, uniform mutation and random replacement operators are used in the algorithm. Random replacement allows Scout agents to adapt their expectation of success under changing conditions of informational and environmental availability.

When dispatching new Scout agents, the User agent uses tournament selection to choose parent genomes from its population, favouring genomes according to the success of the respective Scout agent in locating other satisfactory Scout agents.

The behaviour of each Scout agent depends upon a satisfaction or preference function that indicates the degree of satisfaction that the Scout agent has with an Environment. This satisfaction function employs two bitstrings of length 32, drawn from a binary genome containing 64 bits. These two bitstrings are known as the `XOR_mask` and the `AND_mask`. To determine the degree of satisfaction for a given Environment, the Environment's hashcode is XORed with the `XOR_mask`, and then ANDed with the `AND_mask`. The number of set bits (i.e. bits with the value "1") then indicates the degree of satisfaction with the Environment. This preference function can then be evolved, in order to generate different orderings of preferences for Environments.

Scout agents are initialised with a success of zero. New generations of Scout agents are generated by the recombination of the genomes of two "parent" Scout agents, resulting in two new Scout agent genomes. Two new Scout agents result, that are released into the User agent's local Environment. From this point they carry out a three-phase life cycle (described below). If they complete this life cycle, and return successfully to the originating Environment, an existing member of the population of Scout agents based in that Environment is replaced at random by the genome of the returning Scout agent. In this way Scout agent preferences evolve over many generations in response to the conditions they encounter in different Environments.

## 3.4 Scout Agent Life Cycle

Having been created by User agents in their home Environment, Scout agents go through a life cycle that is divided into three phases: the Exploratory phase, the Sharing phase and the Reporting phase.

In the *Exploratory phase*, a Scout agent visits eight Environments in a random walk starting from the Environment in which it originated. At each Environment it requests four addresses of neighbouring Environments, selecting one of these at random for the next hop. These numbers are fixed across all experiments in order to allow comparison across peer networks of different sizes. After collecting the thirty-two Environment addresses in this way, the Scout agent applies its evolved preference function in order to calculate a satisfaction value for each of the thirty-two potential host Environments encountered. It then selects as a host the Environment with the address that gives it the highest satisfaction. Where several Environment addresses give the same satisfaction, the most recently visited is preferred. The Scout agent then enters the Sharing phase.

During the *Sharing phase* the Scout agent migrates to its preferred host Environment, and spends a pre-determined period interacting with other Scout agents in that Environment - notifying them of its User agent's ID and category of interest, as well as noting the IDs and categories of interest represented by other Scout agents in that Environment. Then it moves to the Reporting phase.

In the *Reporting phase* the Scout agent migrates back to its originating Environment, notifies the originating User agent of its genome, and the number of successful encounters achieved. Scout agent success is measured according to the number of Scout agents encountered that were derived from different User agents (hence different Environments), but represented the same information category. So, a successful encounter in this context means an encounter with a Scout agent originating from other User agents that represent the same information category.

The Scout agent then destroys itself, but its genome may live on in that it may be selected to contribute to the next generation of Scout agents, according to its success in locating Environments that contain other Scout agents representing User agents with common interests. The use of tournament selection means that some Scout agents with success lower than the current Scout agent population average may contribute to the next generation, but they are less likely to, and Scout agents with higher success are more likely to be represented in the next generation. Tournament selection also ensures responsiveness to changing conditions.

### 3.5 Consequences of Agent Behaviour

The repetition of this three-phase life cycle over multiple generations will lead to changes in the numbers of Scout agents found in each Environment at each iteration (corresponding to a generation of the evolutionary algorithm.) The long-term solution should be a network of Scout agents clustered to different densities in different Environments, with average Scout agent preference for Environments evolved to a level that most effectively supports choice of Environments in which agents representing the same category of interest can interact. Accordingly, Scout agent success in achieving such interactions should be maximised. Such a network of information sharing agents may support several distinct groups of agents, as represented by the shaded and unshaded agents shown in Figure 2.

### 3.6 Experimental Conditions

The algorithm described above provided the basis for a series of experiments. In each experimental run we were interested in the effectiveness of the evolutionary learning among agents in stimulating co-location of Scout agents in appropriate Environments. For the sake of logging results, all Environments were hosted in parallel on a single machine. (However, there is no reason why they should not be hosted on multiple machines in parallel in the future.) To compensate for this

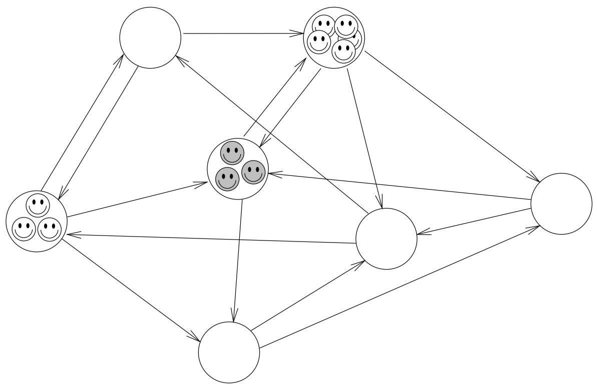

**Fig. 2.** An example configuration of information-sharing agents

lack of true parallelism, User agent search intervals, Scout agent waiting time, and overall run length were made proportionate to the number of User agents. A minute of CPU time was provided for the activity of each User agent. Each User agent began the simulation with a fixed category of interest, and a population of 100 Scout agents with random genomes (defining random preference functions). Initial experiments used the same category of interest for all User agents, but more than one category of interest can be used if required.

## 4 Results

Figure 3 shows the progress of a single experiment, involving thirty-two User agents. The number of Scout agents in each Environment changes over time due to the migration of Scout agents between Environments, as well as being due to the evolution of Scout agent genomes. The evolutionary algorithms are executed in real time by the parallel operations of all the User agents. For this reason results are shown against CPU time.

It is clear that one Environment in particular becomes the most popular, attracting the vast majority of Scout agents in the system. This distribution of Scout agents, with few agents in many Environments, and many in few, is the result of selection of Scout preferences for Environments based on interactions between Scout agents during the Sharing phase of their life cycle. This grouping of Scout agents could then be used to support more effective information exchange among the User agents in the system than was possible at the start of the experiment. It indicates how this evolutionary approach could be useful in facilitating information interactions between the human users who have established such User agents.

Figure 4 shows how the phenomenon shown in Figure 3 occurs. Over time average Scout agent success increases, because the independent evolutionary algorithms converge to common Environmental preferences. This increases the

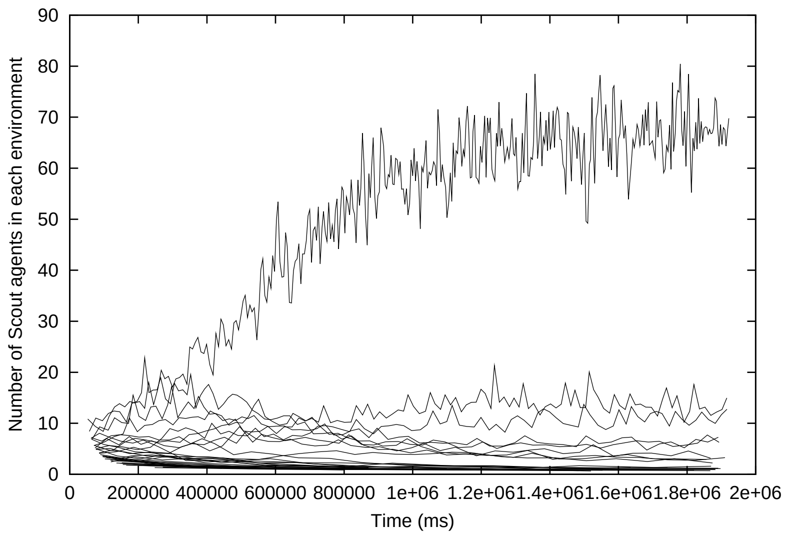

**Fig. 3.** Environment population over time - 32 User agents

Scout agent population density in certain Environments and hence increases Scout success.

If more User agents (and hence more Environments and more Scout agents) are involved, the system takes longer to evaluate where the appropriate Scout agents are, but still identifies them in the end. In Figure 5 we show the results of multiple runs of the algorithm designed to calculate the average (mean) value of Scout agent success. This is calculated after each 1 minute of CPU time has been used per User agent. We are interested to see whether use of the evolutionary algorithm has an effect on the average success of Scout agents.

Figure 5 shows that this occurs, in that average Scout agent success after one minute of CPU time is greater than the initial value (of zero). If the evolutionary algorithm is not used, so Scout agents have uniform preferences, average Scout agent success after one minute of CPU time, although non-zero, is constant irrespective of the number of User agents involved. If the evolutionary algorithm is used (represented by evolved preferences in the Figure), it is interesting that the average Scout agent success actually increases with the number of User agents, before declining at higher numbers of User agents. This suggests the benefit that the use of evolutionary techniques can offer among populations of agents in multi-agent systems, but also implies that very high numbers of User agents may make it more difficult for successful interactions between Scout agents to arise.

The results in Figure 3 shows that the evolution of Scout agent preferences for Environments can support convergence of many Scout agents to a single preferred Environment. When larger numbers of User agents are spread across more Environments, evolution of Scout agent preferences may result in several

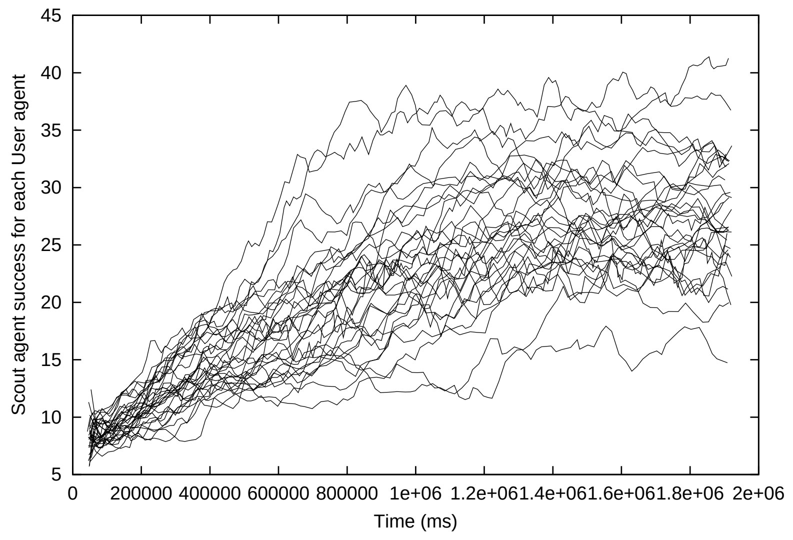

**Fig. 4.** Average Scout success over time - 32 User agents

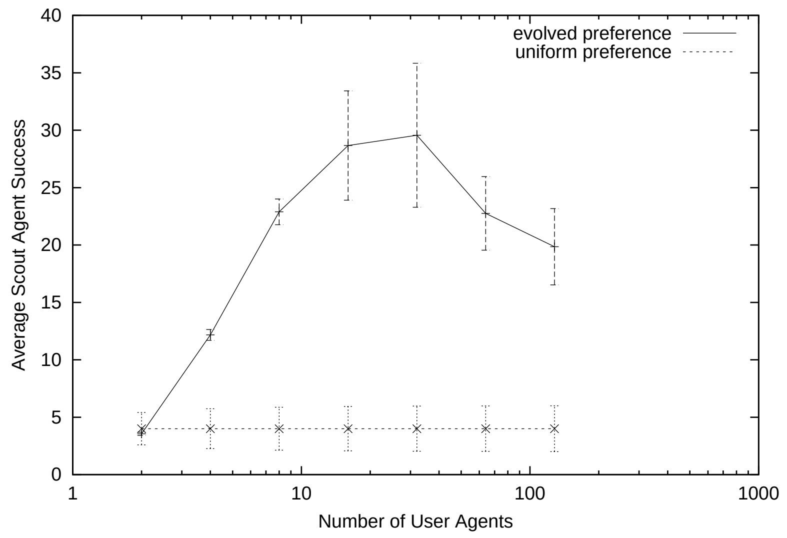

**Fig. 5.** Average Scout agent success after one minute CPU time per User agent

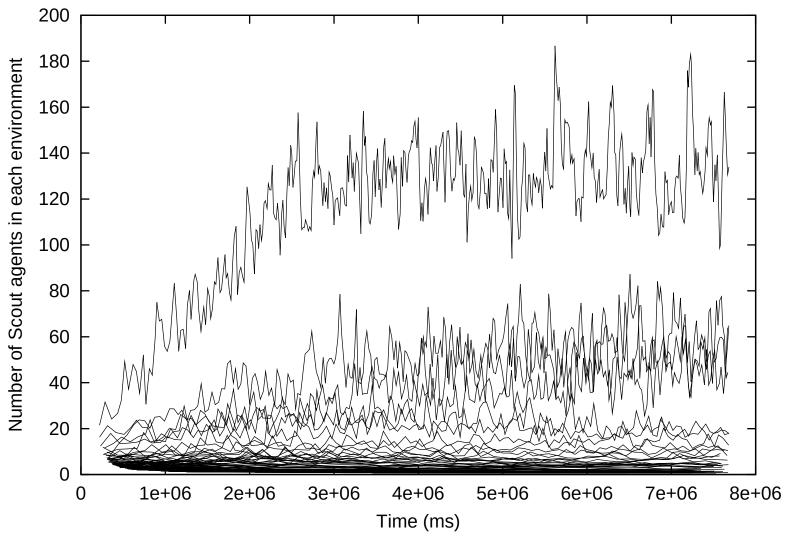

**Fig. 6.** Environment population over time - 128 User agents

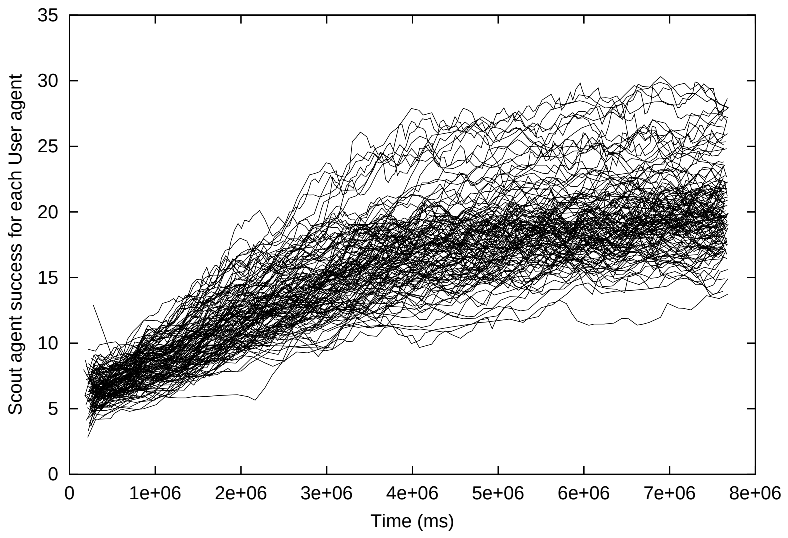

**Fig. 7.** Average Scout agent success over time - 128 User agents

Environments supporting significant numbers of Scout agents in the long term (Figure 6). This does not indicate a failure to evolve to a sufficiently preferred solution, as comparison of the changes in average Scout success over time with this higher number of User agents with Figure 4 shows a similar change in success results although the final average is different (see Figure 7). In fact one Environment in Figure 6 ends up with significantly more Scout agents than all the others after the algorithm is run for some time. But this does not eliminate the several Environments that maintain persistent populations of Scout agents at somewhat lower levels. This is an inevitable source of the use of a random walk by Scout agents in locating Environments.

## 5 Discussion

The experiments in evolving group formation that we have implemented using the DIET platform suggest that evolving agent preferences may be a useful means to tackle information management problems. Starting from a random initial assembly of users, agents quickly co-locate according to the interests of their respective User agents. This facilitates more rapid and effective communications between User agents representing human users with common interests, and so shows the potential for application to more general peer-to-peer collaborative networks [18]. The experiments presented here are designed such that many User agents represent similar interests, but it would be possible to develop alternative scenarios where very many different interests were represented, and Scout agents spread out over many Environments.

While the results given above show convergence of the majority of Scout agents in the system to a single preferred Environment, it is likely that Scout agents will encounter a variety of Environments during exploration. The coexistence of agents in multiple Environments may provide additional robustness, since the loss of specific machines hosting some Environments is unlikely to eliminate all members of a specific agent community in a sufficiently large network. In addition agents persisting in such a diminished system will have the capability to evolve their preferences so as to adapt to the remaining Environments. In fact, because users for the Scouts that converged on a specific Environment that has just disappeared, all have similar evolved preferences, their Scouts are likely to quickly converge on an Environment with a similar hashcode.

The experiments described above implement agents in multiple Environments in parallel on a single machine. It would be worthwhile to investigate larger networks of Environments and User agents with diverse categories of interest. Accordingly, further experimentation is planned, using multiple computers connected in a Beowulf cluster [20]. This should help reduce artefacts arising from thread sharing, and also permit the construction of larger peer networks.

The implementation of preference evolution on multiple machines may provide a means of using this algorithm for load balancing. This is because the use of multiple machines and hence system resources in parallel will provide the agents involved with the potential to evolve preferences and vary success at different rates in different machines. As a consequence, Scout agents will have the opportunity to switch between machines in order to improve their success rate in interacting with other Scout agents. While initial convergence of most Scout agents to a single Environment may result in a similar way to that in the experiments shown here, a consequence of this will be increased demands on one of the machines in the peer network. This will place constraints on the Environments hosted on that machine, restricting agent interactions. This may stimulate migration of Scout agents to other machines where system resources are less heavily in demand. The consequence of this will be a contrasting pressure on Scout agents to disperse over multiple machines, a kind of implicit evolution driven by available system resources.

This implicit evolution could be further used to develop groups of information sharing agents. The DIET platform provides the means to monitor the use of system resources by agents. Accordingly computational resource cost could be used to constrain the evolutionary algorithm so as to develop preferences appropriate to the machines (and/or resources) available at the time. In this way agents adopting computationally efficient behaviour can be evolved without explicit population management.

## Acknowledgements

This work was carried out as part of the DIET (Decentralised Information Ecosystems Technologies) project (IST-1999-10088), within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union. We thank the other participants in the DIET project, from the Departmento de Teoria de Señal y Communicaciones, Universidad Carlos III de Madrid, the Department of Electronic and Computer Engineering, Technical University of Crete, and the Intelligent and Simulation Systems Department, DFKI, for their comments and contributions. We also acknowledge the support of the Enterprise Venturing Programme of BTexact Technologies.

## References

1. Bäck, T., Fogel, D., Michaelewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics (2000)
2. Bonsma, E., Shackleton, M., Shipman, R.: Eos: an evolutionary and ecosystem research platform. BT Technology Journal **18** (2002) 24–31
3. DIET project: web site. `http://www.dfki.uni-kl.de/DIET` (2001)
4. European Commission IST Future and Emerging Technologies: Universal information ecosystems proactive initiative. `http://www.cordis.lu/ist/fethome.htm` (1999)
5. Gallardo-Antolín, A., Navia-Vázquez, A., Molina-Bulla, H., Rodríguez-González, A., Valverde-Albacete, F., Cid-Suerio, J., Figueiras-Vidal, A., Koutris, T., Xiruhaki, C., Koubarakis, M.: I-Gaia: an information processing layer for the DIET platform. In: Proc. 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002). Volume 3., Bologna, Italy (2002) 1272–1279

6. Gordin, M., Sen, S., Puppala, N.: Evolving cooperative groups: preliminary results. In: Proc. of the AAAI-97 Workshop on Multi-Agent Learning. (1997)
7. Haynes, T., Sen, S., Schoenefeld, D., Wainwright, R.: Evolving a team. In: Proc. AAAI Fall Syposium on Genetic Programming, Cambridge, MA (1995)
8. Hoile, C., Wang, F., Bonsma, E., Marrow, P.: Core specification and experiments in DIET: A decentralised ecosystem-inspired mobile agent system. In: Proc. 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002). Volume 2., Bologna, Italy (2002) 623–630
9. Jonker, C., Klusch, M., Treur, J.: Design of collaborative information agents. In: Proc. 4th Int. Workshop on Cooperative Information Agents. Number 1860 in LNAI, Berlin, Springer (2000)
10. Klusch, M.: Information agent technology for the internet: a survey. Journal of Data and Knowledge Engineering (2000)
11. Klusch, M., Sycara, K.: Brokering and matchmaking for coordination of agent societies: a survey. In Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R., eds.: Coordination of Internet Agents: Models, Technologies and Applications, Berlin, Springer (2001)
12. Koubarakis, M., Tryfonopoulos, C., Raftopoulou, P., Koutris, T.: Data models and languages for agent-based textual information dissemination. In: Cooperative Information Agents 2002 (CIA-2002), Madrid (2002)
13. van Lengen, R., Bähr, J.T.: Visualisation and debugging of decentralised information ecosystems. In: Proc. of Dagstuhl Seminar on Software Visualisation, Berlin, Springer (2001)
14. Marrow, P., Koubarakis, M., van Lengen, R., Valverde-Albacete, F., Bonsma, E., Cid-Suerio, J., Figueiras-Vidal, A., Gallardo-Antolín, A., Hoile, C., Koutris, T., Molina-Bulla, H., Navia-Vázquez, A., Raftopoulou, P., Skarmeas, N., Tryfonopoulos, C., Wang, F., Xiruhaki, C.: Agents in decentralised information ecosystems: the DIET approach. In: Proc. of the AISB'01 Symposium on Information Agents for Electronic Commerce, York, UK (2001) 109–117
15. Matarić, M.: Designing and understanding adaptive group behavior. Adaptive Behavior **4** (1995) 51–80
16. Moukas, A., Zacharia, G.: Evolving a multi-agent information filtering solution in amalthea. In: Proc. of Agents '97. (1997)
17. National Institute of Standards and Technology: FIPS PUB 180-I. secure hash standard. `http://www.itl.nist.gov/fipspubs/fip180-1.htm` (2001)
18. Oram, A., ed.: Peer-to-peer: harnessing the power of disruptive technologies. O'Reilly Associates, Cambridge MA (2001)
19. Sarma, J., Jong, K.D.: Generation gap methods. In Bäck, T., Fogel, D., Michaelewicz, Z., eds.: Handbook of Evolutionary Computation, Bristol, Insititute of Physics (2000)
20. Sterling, T., Becker, D., Savarese, D., Durband, J., Ranawake, U., Packer, C.: Beowulf: a parallel workstation for scientific computation. In: Proc. 24th Int. Conf. on Parallel Processing. Volume 1. (1995) 11–14
21. Wang, F.: Self-organising communities formed by middle agents. In: Proc. 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002). Volume 3., Bologna (2002) 1333–1339
22. Waring, R.: Ecosystems: fluxes of matter and energy. In Cherrett, J., ed.: Ecological Concepts, Oxford, Blackwell Scientific (1989)