Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System

Cefn Hoile, Fang Wang, Erwin Bonsma, Paul Marrow
Intelligent Systems Laboratory, BTexact Technologies
Antares 2 PP 5, Adastral Park
Ipswich IP5 3RE, UK
+44-1473-642405
{cefn.hoile|fang.wang|erwin.bonsma|paul.marrow}@bt.com

ABSTRACT

Mobile Agent systems have attracted considerable attention as means of exploring and manipulating distributed information sources. However, many existing multi-agent platforms present limitations in terms of adaptability and scalability, indicating difficulties when trying to replicate these results on a large scale. We describe the core of a novel mobile agent toolkit known as DIET, (Decentralised Information Ecosystem Technologies), which addresses some of these limitations and provides a foundation for an open, robust, adaptive and scalable agent ecosystem. We introduce DIET core features and describe how they support basic mobile agent capabilities such as migration and real-time interaction. We then illustrate how an ecosystem-inspired design approach differs from conventional design approaches. Finally, we experiment with a simple information retrieval scenario, demonstrating the emergence of agent communities through the evolution of environmental preferences. In this way we hope to clarify how applications built on this foundation could be used to tackle problems in adaptable and open real-world scenarios.

Categories & Subject Descriptors: I.2.11 Distributed Artificial Intelligence, C.2.4 Distributed Systems, D.1.3 Concurrent Programming, I.2.6 Learning

General Terms: Algorithms, Management, Measurement, Performance, Design, Reliability, Experimentation.

Keywords: Agent, Multi Agent System, Mobile Agent System, Evolution, Group, Community, Ecosystem, Emergence

1. INTRODUCTION

Multi-agent systems have the potential to support complex real-world applications in an open and extensible manner. However specific implementations can often present problems of scalability, robustness and adaptability [29]. The DIET (Decentralised Information Ecosystem Technologies) project [3] draws on analogies with ecosystem dynamics in an attempt to address these limitations in recent work on multi-agent systems. The main goal of the project is to implement and validate an agent development toolkit supporting complex, real world applications, such as the management of information sources like the Internet or large scale intranets.

This paper introduces the core of the DIET toolkit, developed as part of the DIET project. It also presents the results of a simple experiment, built upon features of the core, in which agents adapt their preferences to select execution environments most suitable for their information retrieval needs.

1.1 Adaptivity and Scalability of Multi-Agent Systems

Agent systems have attracted a great deal of interest and have been focused on a wide variety of applications. This has led to the construction of many different agent development toolkits. Examples include ADEPT [16]; JAFIMA [7]; OAA [9]; RETSINA [22] and ZEUS [14].

ADEPT provides an agent-based workflow technology that combines distributed object platforms with software agents to manage business processes [16]. JAFIMA by contrast is a comprehensive agent design framework for developers to develop agents from scratch. It uses a layered agent architecture pattern to form centralised cooperation and translation and shared mobility layers [7]. OAA (Open Agent Architecture) provides an agent architecture that can facilitate communication and cooperation between agents by providing one or more facilitators [9]. OAA is structured in an attempt to minimise the effort involved in creating new agents and wrapping legacy applications. RETSINA is a distributed collection of heterogeneous agents that develops agent collaboration in information retrieval and integration tasks in a reusable way [22]. ZEUS is another comprehensive agent building toolkit that provides a library of agent-level components and an integrated environment to design agent processes [14]. It combines many established technologies, such as agent communication language, reasoning, planning, ontology and visualisation, to provide an integrated collaborative agent building environment.

The above agent development frameworks, amongst many others (e.g. [20]), have greatly improved the study and realisation of collaborative multi-agent approaches and have been applied to various application areas such as information management, distributed resource allocation, and e-commerce. However, these and other multi-agent systems have suffered to a greater or lesser degree because of their closed nature in information discovery, communication, ontology, reasoning, coordination and monitoring, as identified by Nwana and Ndumu [15].

AAMAS'02, July 15-19, 2002, Bologna, Italy.

Some multi-agent systems, such as OAA and JAFIMA, involve much centralised design. This may incur a bottle-neck problem as the agent population, the communication load and the information in the system grows. Other systems like RETSINA and ZEUS, are decentralised architectures in which distributed, heterogeneous agents collaborate in performing tasks together. However, even in agent systems with decentralised features the core implementations of 'white pages' and 'yellow pages' services is often centralised or suffers from scalability problems as more agents and hosts are added. Some of these issues are addressed by RETSINA in [27].

The co-ordination of agent systems often requires prior knowledge from domain developers to carefully define the functionality of individual agents, allowing the agent system to deliver its functionality. Preconditions and post-conditions, for instance, are key to the functioning of a rule-based planner in ZEUS. In addition, each ZEUS agent coordinates with other agents according to its predefined knowledge of others through known protocols and interaction strategies. Multi-agent systems designed in this way are able to function in a well-understood world. But it is questionable whether predefined knowledge and methods are sufficiently flexible when the environment is dynamically changing. Such implementations can also be very susceptible to failure. If some agents responsible for certain tasks are inaccessible or information is unexpectedly absent, agents may be unable to solve the problems themselves, and help must be sought from the outside, i.e., the human designer.

In order to work properly, some systems assume an oversimplified, static, deterministic world, whereas most natural worlds are complex, dynamic, and uncertain [19]. In a complex, dynamic real world context like the Internet or an intranet, for example, information, computational resources and user presence are transient in wholly unpredictable ways.

In an open agent system, it is impossible for developers to anticipate agent behaviours, track all possible eventualities and design corresponding optimal strategies beforehand. Sense, Plan, Act cycles which are dependent on global models are unsatisfactory; as the scale of the network increases, models cannot be kept up to date, and planning becomes increasingly intensive. For these reasons, multi-agent systems relying heavily on the predictability of system behaviour or rational agent behaviour cannot adjust themselves to dynamic changes, scale to radically increased number of users and pieces of information, or be robust to unexpected behaviours or failures in the system.

1.2 The DIET Philosophy

The DIET project [3] is concerned with the development of an ecosystem-inspired approach to the design of agent systems [8], in the hope of addressing some of the limitations mentioned above. In this context an ecosystem can be viewed as an entity composed of one or more communities of living organisms, in which organisms conduct frequent, flexible local interactions with each other and with the environment that they inhabit [23]. Although the capability of each organism itself may be very simple, the collective behaviours and the overall functionality arising from their interactions exceed the capacities of any individual organism. These higher level processes can be adaptive, scalable and robust to changes in their environments [26].

Ecosystems have been a source of inspiration to a number of previous developers of agent systems [e.g., 4, 12, 18]. Moukas [12] employs ecosystem inspired ideas in the *Amalthaea* architecture for information filtering. Van Parunak and colleagues [18] have advocated a "Synthetic Ecosystems" approach to the design of multi-agent systems, using the interaction of many simple agents to solve problems, rather than sophisticated processing at the individual agent level. Doran [4] provides a contrasting view of ecosystem-inspired agent systems, using agent models to assist in the management of natural ecosystems.

Inspiration from ecosystems is the focus for a number of ongoing research collaborations. The information ecosystem concept [5] draws a parallel between the emergent complexity of natural ecosystems and the networks making up the global information infrastructure (e.g., the Internet and Worldwide Web). The DIET project aims to implement an information ecosystem populated by information agents (or infohabitants, after [5]) drawing upon the properties of natural ecosystems mentioned above.

We intend to exploit the dynamics of natural ecosystems, composing applications from simple agents, distributed across many execution environments. In each environment large numbers of asynchronously executing software agents may be created and maintained, (over 500,000 addressable DIET agents can occupy the memory of a modest desktop PC). Agents share the environment's resources and interact together in a way that loosely parallels the relationship of biological organisms with their natural environment. In particular, they are capable of autonomous migration between environments, providing a basis for the dynamic creation of 'ecological' communities. These communities can be heterogeneous and interdependent, and are autonomously selected by participating individuals.

Agents can conduct communications with each other, exchanging information or resources to deliver application objectives and carry out system functions. We focus on the implementation of lightweight agents with very simple behaviours, rather than emulating the complex behaviours of sophisticated organisms like human beings. DIET agents are lightweight and can respond rapidly to new information with minimal processing. In this way, the DIET platform provides a foundation for global information management through the emergent effects of local cooperation, offering a basis for ecosystem-inspired experimentation.

The rest of the paper is structured as follows. The next section describes how the DIET philosophy has been put into practise. We outline the architecture of the DIET core and identify some of its key functions and components, before discussing in more detail an example of its functionality - the communication facilities between DIET agents. Section 3 describes an experiment in which agents evolve habitat preferences in a transient Peer-to-Peer network, improving the efficiency of information sharing. Finally we discuss how these results illustrate the potential of the DIET platform to overcome common limitations of multi-agent systems in application to real-world problems.

2. IMPLEMENTATION

2.1 Architecture

DIET applications can be decomposed into three separate layers providing an extensible framework for the exploration of ecologically-inspired software solutions [8];

Core layer : The functionality supported by the lowest layer is deliberately minimal as will later be explained in more detail.

ARC layer : Additional utilities are distributed along with the core, known as ARC or 'Application Reusable Components'. These provide primitives which exploit the minimal functions of the core to support higher level activities common to many applications. These include remote communication, multicasting and directory services.

Application layer : This layer comprises additional data structures and agent behaviours for application-specific objectives.

2.2 Core Layer Functionality

Like other Multi-agent systems a fundamental unit in any DIET application is the *Environment*, which provides a location for agents to inhabit and a number of services to support agent activities. Each Java Virtual Machine [JVM] contains a single '*World*' containing one or more environments. In turn, multiple Worlds may run on a single computer, and multiple computers may run concurrently within a network. The construction of the core deliberately anticipates the scaling of the DIET 'Universe' to an indefinite number of agents, environments and worlds.

The minimal functionality provided by the environment consists of: 1) agent creation; 2) agent destruction; 3) agent migration; and 4) local communication. These four key functions are executed with minimal overhead (e.g. memory, processor and network overhead). Specifically, the CPU load of each primitive service is not systematically dependent on the number of individuals occupying the environment, allowing environmental service requests to return rapidly even in well-populated environments. This allows efficient and scalable operation of the system.

In many MAS projects the majority of processing is expected to take place through individual agents executing decision algorithms. By contrast, DIET information processing is expected to take place through the interactions between simple agents, demanding highly efficient implementations of the core services. We believe that these four operations are the minimum required for a mobile agent system, and that any further functionality required can be efficiently built on this foundation.

2.3 Communication

The impact of the design philosophy and organisational structure of the DIET agent system is well illustrated by examining the support for agent communication offered by the different layers.

Local communication is offered as a core layer service. This function is implemented as a direct connection between agents in the same environment. Although this core functionality is minimal, extra services, such as remote communication across different environments, can be delivered as a service by specialised agents, using only the four 'core' functions above.

2.3.1 Local Communications – A Core Feature

The core layer provides support for communication only between local agents. We define local communication as the passing of messages and objects between two agents occupying the same environment. This implies that they occupy the same machine. The following sections describe briefly the issues considered in our implementation.

2.3.1.1 Identities and Indexing

In order to start communicating, the initiating agent must first specify the target individual with which it wants to interact. An agent can be identified in one of two ways.

Each agent is assigned a unique binary 'name tag' randomly generated within its originating environment. Given a random bitstring of sufficient length, identity clashes are incredibly unlikely. Hence this unique identifier can be employed to distinguish a single agent throughout its lifetime, and it is retained following migration to different environments.

In addition, on startup, each agent may select a 'family tag' of its own choosing. It is expected that this tag would be used to reflect the services that it offers. Duplication of family tags is expected. Where more than one individual of the requested family exists, one of them is returned at random.

The decentralised naming approach in the core platform avoids the limitations inherent in centralised, synchronised name servers. No forms of universal addressing, wildcard indexing or exhaustive lists are supported, ensuring the rapid return of connection requests with a CPU cost on average independent of the number of occupants. This also ensures the scalability of the DIET 'universe' to indefinitely many machines - only local indexes exist in each machine, allowing the rapid retrieval of a single agent within its own environment by either name tag or family tag. If no appropriate agent exists in the environment, the service returns rapidly with a failure code. Rapidly-executing synchronised code allows the integrity and accuracy of the environment's local indexes to be maintained under all conditions, and minimises delays to agents sharing the index.

2.3.1.2 Connections

Once the target agent is identified, a '*connection*' is created allowing the two individuals to dispatch messages asynchronously into each others' buffers. There is a superficial advantage to the provision of limitless message buffers since agents do not need to handle message failure. However, without memory constraints, the system may crash unexpectedly. Furthermore, an agent with a large message backlog cannot respond in a timely fashion to incoming messages, reducing its effectiveness in a real-time system. For these reasons message buffers are deliberately constrained. The idea of putting a strict limit on certain resources is used throughout the DIET core. It makes high system load visible to agents, encouraging them to modify their strategies in response to prevailing conditions.

Instant feedback is always available to determine whether a given connection is alive (the agent exists), and whether a communication has completed successfully (the message has been successfully placed in the target buffer). Once a connection has been created, direct memory references can be used to avoid any form of lookup, eliminating the bottleneck of synchronised indexes. Long term connections can thus be retained between commonly interacting individuals, minimising costs. When either agent migrates or dies, local connection records permit automatic notification of the other agent, allowing it to modify its behaviour.

To maximise the efficiency of agent strategies, a transaction state object known as a 'Context' is made available whenever a message arrives or disconnection event takes place. A distinct 'Context' is associated with each participant in a connection. Context objects may take whatever form the agent designer feels

appropriate to maintain transaction state. They offer an efficient means of handling multiple transactions concurrently, since they are directly referenced by the connection, also avoiding lookup.

In addition to communication of local agents, exploiting decentralised information and computational resources depends upon information exchange between DIET environments. In order to achieve this, remote communication is required.

2.3.2 Remote Communication – An Extended Service

A variety of mobile agent addressing strategies can be explored to achieve remote communication, including centralised indexes, forwarding pointers, home agents and other agent tracking strategies [11]. However, there is no explicit support for such strategies in the core. Instead, the DIET system provides multiple forms of remote communication through ARC and application layer extensions, exploiting a variety of strategies to suit different applications and scenarios. These forms of remote communication include Carrier Pigeon, Mirror and Mirror Provider.

Simple, one-off remote communication can be implemented by creating a 'Carrier Pigeon' agent which migrates to a remote environment and employs local communication to deliver a message to the addressee.

The complexity of managing 'Carrier Pigeons' can be eliminated by the creation of a local 'Mirror' agent. Mirrors provide a communication channel to a remote individual, and employ Carrier Pigeons to achieve their function. An agent in environment A can interact with a local mirror as if it were the individual in environment B. The individual in B in turn interacts with a local mirror as if it were the agent in environment A. In this way, remote communication is transparent to both participants. Furthermore, it provides a level of indirection in the creation of a Carrier Pigeon. Individuals are no longer responsible for creating Carrier Pigeons directly. Thus, alternative Carrier Pigeons with alternative behaviours may be substituted at runtime to improve the efficiency or reliability of remote communication.

The use of a Mirror is a very flexible arrangement, demanding very little of the participants. The mirrored communication can be initiated by either agent, or even by a third agent if necessary, catalysing interaction between agents in different environments for their own ends. Intermediation of the network transport layer is discernable only implicitly, through the delays between dispatch and response.

In addition, Mirror Providers may be introduced which construct Mirrors at the request of an agent. In this way, the exact behaviour of the Mirror used may be exchanged without the knowledge of the initiating agent. The choice of underlying network transport, and strategies for handling failure modes may be modified. This form of soft addressing allows very simple agents to benefit from the latest innovations available, without explicit awareness of the range of implementations.

All 'Carrier Pigeons', 'Mirror Agents' and 'Mirror Providers' are included within the ARC layer of the core distribution. Further service agents are also available at this stage, including multicasters which handle the complexities of multicast communications, directory agents which allow arbitrary information to be publicised to interested agents, and other fundamental services.

2.3.3 Generalising Service Provision by Agents

The above sections introduced the design philosophy behind the DIET platform, i.e. how to design and implement fundamental functions (e.g. local communication) and how to extend these functions to achieve higher level services (e.g. remote communication). The idea of efficient, robust, scalable and flexible implementation runs through the whole platform. Developers may choose to employ primitives offered as part of the DIET distribution, or create new implementations that are better suited to their purposes. However, the differences between implementations do not need to be *explicitly* externalised in the form of metadata. Instead, complex, manifold functions may arise from different combinations of simple agent behaviours. The experiment detailed in the next section indicates one way in which agents can select clients, peers or service providers through *implicit* performance measures.

3. EVOLVING GROUP FORMATION

To illustrate how the DIET system can be applied to information ecosystems, we describe here an experiment into group formation within the DIET world, which uses an evolutionary algorithm to evolve each individual's habitat preferences, hence implicitly choosing the combinations of peers with which it interacts in order to improve its performance in information retrieval.

The dynamic formation of communities of agents could be very important for the proper exploitation of computational and informational resources in future networks. The mutual relationships formed by organisms in nature have been characterised by some as the primary influence in increasing adaptive complexity over evolutionary time [10]. In nature many partnerships arise without the rational negotiation which is characteristic of classical agent-based systems. Group formations simply persist and propagate where their collective strategy is competitive with other individuals and groups in their environment. No metadata or centralised coordination is required.

The most rapid and efficient interactions between agents are those that take place locally, between agents occupying a single environment. Occupying a common environment can thus provide advantages analogous to those gained by firms forming industrial clusters [1]. Examples include: improved communication efficiency; improved access to directory and other system services; specialisation of local services, and more rapid information propagation. The following experiment is intended to illustrate these advantages.

3.1 An Information Sharing Network

The scenario for experimentation is focused on the formation of collaborative information-sharing communities. We imagine a peer to peer network of transient users connecting to the network in order to identify resources which satisfy their information needs. Each user has an associated 'category of interest'. Users of the same category are interested in the same set of information, but initially only have access to a subset. Thus, it is in their interest to find other users of the same category and share information with them. Each user creates a DIET environment, within which Scout agents can be created. The agents then migrate through the peer network in search of other Scouts serving users of the same category.

When a new user connects a locally running DIET environment to the peer network, both the availability and the demand for computational resources increases. Availability and demand in the network as a whole can be considered proportional to the number of participants in the peer network.

It is important, as previously mentioned, for agents that interact frequently to be co-located. However, since there are no permanent servers in the network, (in a peer network all environments may be transient), it is not possible to decide upon a permanent home address for specific user groups, (groups denoted by a category of interest). In this experiment, we exploit the evolution of habitat preference to permit agent populations to accommodate these changing circumstances.

3.1.1 Scout Life Cycle

The Scout life cycle is divided into 3 phases: the Exploratory phase, the Sharing phase, and the Reporting phase.

In the *Exploratory phase*, a scout visits 8 environments in a random walk, and requests 4 neighbouring addresses from each environment, selecting at random for the next hop. These numbers are fixed for all experiments to allow comparison across peer networks of different sizes. After collecting the 32 environment addresses, (some of which may be duplicates), the scout applies its preference function to calculate a satisfaction value for each of the 32 potential hosts encountered (each scout's preference function is based upon an evolved bitstring – its genome - as explained later). It then selects a host - the environment address that gives it the highest satisfaction. Where several addresses have the same satisfaction, the most recently visited environment is preferred. The scout then enters the Sharing phase.

During the *Sharing phase*, the scout migrates to its preferred host, and spends a pre-determined period interacting with other scouts in the environment - notifying them of its user's id, and his/her category of interest, and noting the others' ids and categories. Then it moves to the Reporting phase.

In the *Reporting phase*, the scout returns to its originating environment, notifies the user of its genome, and the number of successful encounters achieved. Although the scout destroys itself, its genome survives, and may be selected as a parent in the future according to its success. *Scout Success* is measured according to the number of Scouts encountered that belonged to different users, but represented the same information category.

3.2 Habitat Preferences

3.2.1 Preference Function

Each environment in the peer network has a distinctive signature - a hashcode, generated based on the environment's address within the DIET World. We currently use a 32 bit hashcode, since a unique hashcode of this form can be acquired from all Java objects. However, this form of hashcode could be replaced by one of many different hashing schemes if required e.g. [13].

The satisfaction function of Scouts employs two bitstrings of length 32, (drawn from a binary genome containing 64 bits), 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, then ANDed with the AND_mask. The number of set bits (i.e. bits with value "true"), then indicates the degree of satisfaction with the environment.

3.2.2 Evolutionary Algorithm

Each User manages an independently evolved population of scout genomes. Scouts are bred by Users to maximise Scout Success via a Steady State Genetic Algorithm [21]. A scout's preference for hosts is determined by a bitstring 'genome' provided at birth. In this way scout preferences evolve over many generations in response to the conditions they encounter in different environments. The operators employed to evolve the populations of Scout agents, drawn from the Eos evolutionary and ecosystem platform [2], were tournament selection, two-point crossover, uniform mutation and random replacement.

When dispatching new Scouts, the User uses tournament selection to choose parent genomes from his/her population, favouring genomes according to 'Scout Success'. Individuals are initialised with a success of zero.

The parents' genomes are recombined, creating two new genomes. These new scouts are released into the user's local environment, beginning the three phase life cycle described earlier. If they complete their life cycle, and return successfully, an existing member of the population is replaced at random by the genome of the returning scout.

3.2.3 Distribution of Scouts

Figure 2 shows a illustration of what a resulting information sharing network could look like. In this network, there are ten scout agents representing two categories of interest. The edges of

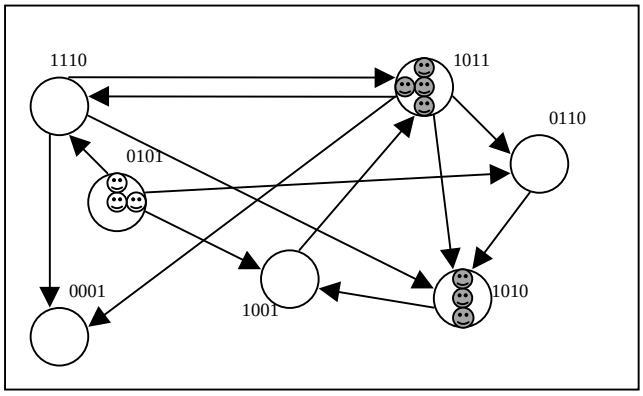

Figure 2: An example of an information sharing network

the graph represent neighbourhood links between environments, provided as an explicit feature of the DIET core implementation.

In this information sharing network, the unshaded scout agents have chosen to cluster at the environment with hashcode 0101. The shaded agents preference is equivalent for both environments 1011 and 1010, hence they distribute themselves evenly.

The relaxed preference of the unshaded group may be to their advantage, since the processor load of a less populated environment can help them execute their information search more rapidly. However, there is a tradeoff since there are fewer agents with which to share information in less populated environments. One of the benefits of using evolution, and selecting according to Scout performance is that these tradeoffs can be optimised implicitly, according to the performance of each individual agent, without central management.

3.2.4 Emulating User Behaviour

Each User hosts a single DIET environment, enabling the creation and dispatch of his/her own Scout agents into the peer network to retrieve information, as well as providing an environment for interactions between incoming Scouts.

In constructing the peer network, pairs of environments are chosen at random and directed neighbourhood links are created between them, permitting agents to transit between the selected environments. This is intended to reproduce the connectivity of a fully decentralised peer network. On average, each has 4 links.

A series of experiments were run, testing the behaviour of the algorithm for a variety of network sizes. To date, it has been tested for 2, 4, 8, 16, 32, 64 and 128 users. For the sake of logging, all environments were actually hosted in parallel on a single machine. To compensate, user search intervals, scout waiting time, and overall run length are proportional to the number of users. We provide a minute of CPU time for each user's activities - a specification consistent with the peer to peer scenario of Section 3.1. Each user begins the simulation with a fixed category of interest, and a personal population of 100 Scouts with random genomes, (defining random preference functions).

3.3 Results

To examine the behaviour of evolving preferences, we have taken logs of asynchronous agent interactions within a DIET system.

The purpose of the experiment was to demonstrate that agent behaviour could be tuned to improve performance in a real-time application scenario. The population in each environment, (*number of Scouts*), against time (*milliseconds*) is shown in Figures 3 to 5 for typical experiments. The average Scout Success for each User, (*number successful interactions*), against time (*milliseconds*) is shown in Figures 6 to 8 for typical experiments

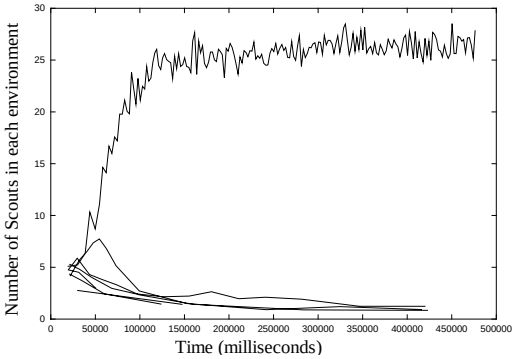

Figure 3 Environment population over time: 8 Users

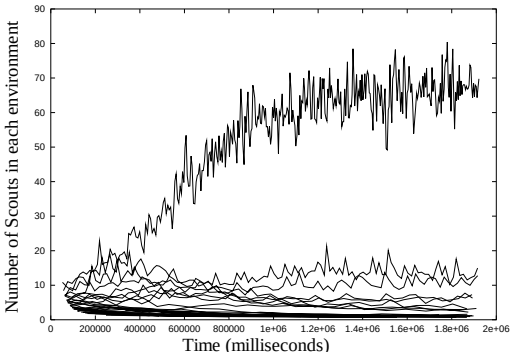

Figure 4 Environment population over time: 32 Users

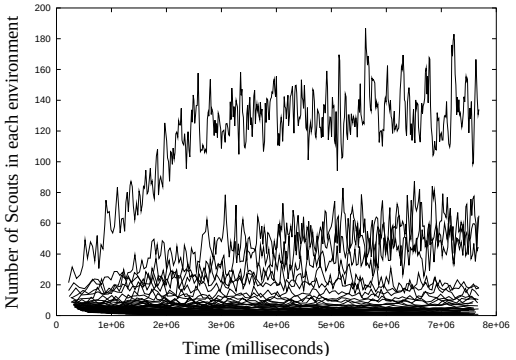

Figure 5 Environment population over time: 128 Users

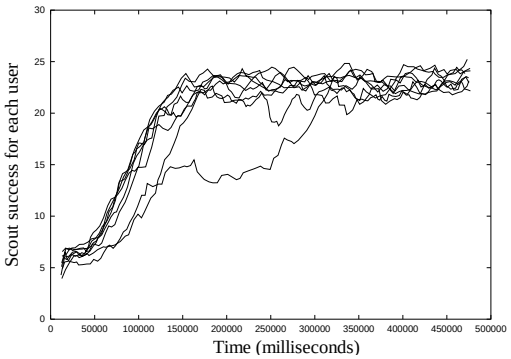

Figure 6 Average Scout Success over time: 8 Users

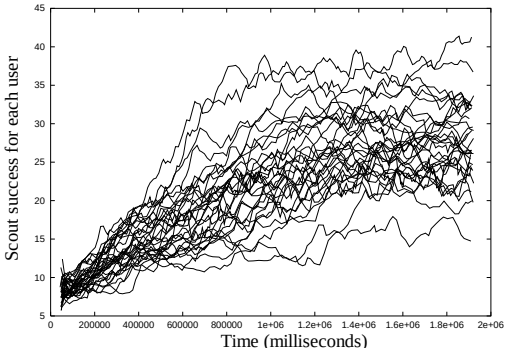

Figure 7 Average Scout Success over time: 32 Users

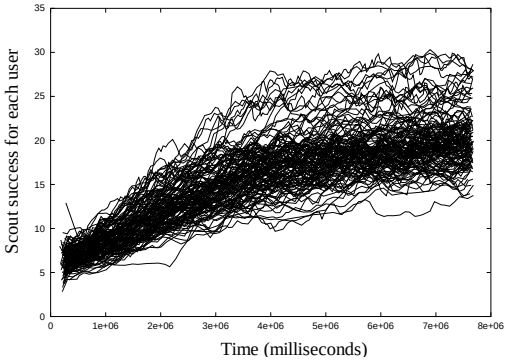

Figure 8 Average Scout Success over time: 128 Users

Finally, the plot in Figure 9 shows the average scout success level (y-axis linear scale) against the number of users (x-axis, log scale). These results are averaged over 10 experiments.

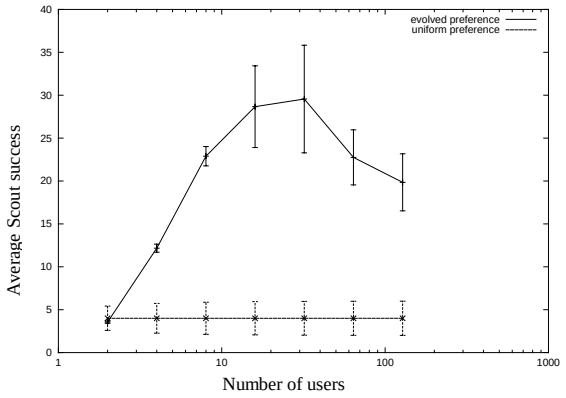

Figure 9 Average Scout success after 1 min CPU per User

4. DISCUSSION

In the cases shown, (Figures. 3 to 9), all Users are of the same information category. That is, all users benefit from sharing with all other users. This simplifies the scenario and makes the results of convergence easier to interpret. In a more diverse agent system, more than one population can develop its own specialised "industrial cluster", converging on environmental preferences shared only with members of its own community - leading to load balancing across multiple environments.

As can be seen from the plots of Scout population, (Figures 3 to 5) the randomised scouts adopt a roughly uniform distribution across all available environments, at the start of each run. However, as time goes on, Scout's host preferences converge on a single environment, or a small subset of environments.

This phenomenon can be straightforwardly explained as a case of positive reinforcement. If the environment selected by a Scout hosts many other Scouts of the same category, it will be more successful in its information sharing mission. Choosing a popular environment is the only way to achieve a selective advantage, favouring scouts whose preferences intersect with other scouts of the same category.

Minor asymmetries arise naturally from the stochastic processes and real-time artefacts in the system. Once an asymmetry exists, this asymmetry is amplified through positive reinforcement as the sexually independent populations experience increasing density within specific environments. For this reason, Scout populations whose Users have common interests will converge to common environment preferences.

As the number of users in the system grows, the scouts distribute themselves over a larger number of environments as in Figure 5. This is explained when we recall that scouts in all experiments sample 32 environments before choosing a host. In the 8-user case, it is likely that scouts will encounter all 8 environments during exploration. Readers may note that in this case a single environment is independently selected by all co-evolving scout populations as the preferred environment. However, as the number of environments increases, it becomes increasingly likely that scouts will only encounter a subset of available environments and users. Despite this limited knowledge, preferences still converge strongly from the initial uniform distribution.

The co-existence of multiple communities may actually provide additional robustness, since the loss of specific machines, or groups of machines is unlikely to eliminate all members of a specific agent community in a sufficiently large network. Load balancing could also be implicit in this behaviour - data suggests that not all Scouts will choose the same host in a large network.

The key indicator is, of course, the improvement in Scout performance. A strongly positive trend in scout success is clear in the experimental scenarios (Figures 6 to 8). It is crucial to examine how this benefit scales with the user population.

However, the results shown in Figure 9 raise some concerns. There is a noticeable downturn in Scout performance as the number of Users increases. It is possible that competing conventions for environment selection remain for longer, since less information is available about global host adoption for a given category. This may prevent convergence on a single environmental preference in the period of time allowed. Other influences may be responsible for this tailoff, such as the overhead of scheduling or synchronising up to 1000 simultaneously executing agents in a single machine.

Further experimentation will be undertaken shortly using multiple computers (in a 32-processor Beowulf cluster [24]), to implement true parallelism. This could help reduce artefacts from thread scheduling, and also permit the construction of larger peer networks. The use of the cluster further allows the robustness of the DIET core and the evolved preference strategy to be tested by arbitrary machine shutdowns.

The experiments in evolving group formation that we have implemented in the DIET platform further confirm its suitability for providing scalable and adaptive solutions to information management problems. Starting from a random initial assembly of users, agents quickly and efficiently group according to the interests of their respective users. This facilitates more rapid and effective communications between users with common interests, and so show the potential for application to more general peer-to-peer collaborative networks [17].

Finally, this application offers an ideal use case for the DIET visualisation toolkit, currently under development at DFKI, which is intended to support the online exploration of complex agent interactions [28].

5. CONCLUSIONS

In this paper we have identified some shortcomings of existing agent systems in terms of lack of scalability and adaptability, and so provided a justification for the construction of a novel toolkit, the DIET platform, that is lightweight, scalable and decentralised.

The design philosophy and several specific features of the DIET core have been introduced, with a focus on the generality of the implementation, and the extensibility and flexibility of service provision. Examples have been provided indicating the ways in which the minimal core functionality is extended through the contribution of specialist agents.

We have shown through experiments that evolution can be used to co-ordinate cooperative interactions between self-interested agents in fully decentralised, transient peer-to-peer information sharing networks. In particular, we have shown that groups of agents with common interests, akin to "industry clusters" [1], can emerge by evolving agents' host preferences without any explicit planning, or any centrally agreed locations.

We are confident that the DIET agent toolkit offers a substantial experimental foundation for ecologically inspired algorithms in distributed information processing.

6. 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 Departmento de Teoria de Señal y Comunicaciones, 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 support from the Enterprise Venturing Programme of BTexact Technologies.

7. REFERENCES

[1] Arthur, W. Urban Systems and Evolution, in Casti, J.L. & Karlqvist, A. (eds.) *Cooperation and Conflict in General Evolutionary Processes*, Wiley, 1994

[2] Bonsma, E., Shackleton, M. & Shipman, R. Eos – an evolutionary and ecosystem research platform. *BT Technology Journal* 18(4), 24-31, 2000

[3] DIET Project web site:

http://www.dfki.uni-kl.de:8080/DIET/public/index.html, 2000

[4] Doran, J. Intervening to Achieve Co-operative Ecosystem Management: Towards an Agent Based Model. *Journal of Artificial Societies and Social Simulation* 4(2): http://www.soc.surrey.ac.uk/JASSS/4/2/4.html, 2001

[5] European Commission IST Future and Emerging Technologies, Universal Information Ecosystems Proactive Initiative, http://www.cordis.lu/fetuie.htm, 1999

[6] Huberman, B.A. *The Ecology of Computation*, North Holland 1988

[7] E.A. Kendall, C.V. Pathak, P.V.M. Krishna, and C.B. Suresh, The Layered Agent Pattern Language, *Proceedings of the Conference on Pattern Languages of Programs* (PLoP'97), 1997.

[8] P. Marrow, M. Koubarakis, R.H. van Lengen, F. Valverde-Albacete, et al., Agents in Decentralised Information Ecosystems: the DIET Approach, *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce* York, UK 2001, pp.109-117.

[9] D.L. Martin, A.J. Cheyer, and D.B. Moran, The Open Agent Architecture: a Framework for Building Distributed Software Systems, *Applied AI Journal*, 13(1/2), 1999.

[10] Maynard Smith, J. and Szathmary, E. "The Origins of Life" OUP, 1999

[11] Moreau, L. Distributed Directory Service and Message Router for Mobile Agents. *Science of Computer Programming*, 39(2-3):249-272, 2001.

[12] Moukas, A. Amalthaea: Information Discover and Filtering using a Multiagent Evolving Ecosystem. *Proc. Conf. Practical Applications of Agents and MultiAgent Technology*, 1997.

[13]National Institute of Standards and Technology FIPS PUB 180-I: Secure Hash Standard http://www.itl.nist.gov/fipspubs/fip180-1.htm

[14] H.S. Nwana and D.T. Ndumu, A Perspective on Software Agents Research, *The Knowledge Engineering Review* 14(2), 1999, pp.1-18.

[15] H.S. Nwana, D.T. Ndumu, L.C. Lee, and J.C. Collis, ZEUS: a toolkit for Building Distributed Multi-agent Systems, *Applied AI Journal*, 13(1/2), 1999, pp.129-185.

[16] P.D. O'Brien and M.E. Wiegand, Agents of Change in Business Process Management, *Lecture Notes in Artificial Intelligence*, v1198, 1997, pp.132-145.

[17] Oram, A. (ed.) *Peer-to-peer: harnessing the power of disruptive technologies*, O'Reilly Assoc., 2001.

[18] Parunak, V., Sauter, J., Clark, S. 1997 Toward the Specification and Design of Industrial Synthetic Ecosystems. *Fourth International Workshop on Agent Theories, Architectures, and Languages* (ATAL'97).

[19] L. Pryor, Decisions, Decisions: Knowledge goals in planning, *AISB Quarterly 92*, Summer, 1995

[20] Ricordel, P.M. & Demazeau, Y., From Analysis to Deployment : a Multi-Agent Platform Survey. In: *Engineering Societies in the Agents' World (ESAW'00), ECAI 00*, Springer Verlag, pp. 93-105, Berlin, August 2000.

[21]Sarma, J. & De Jong, K. Generation gap methods. In: *Handbook of Evolutionary Computation*, C2.7, Bäck, T., Fogel, D. & Michaelewicz, Z. (eds.), Institute of Physics, 2000.

[22] K. Sycara, A. Pannu, M. Williamson, and D. Zeng, Distributed Intelligent Agents, *IEEE Expert* 11(6), 1996, pp.36-46

[23] P.H. Waring, Ecosystems: Fluxes of Matter and Energy, in: *Ecological Concepts*, J.M. Cherrett (ed.), Blackwell, 1989.

[24] Sterling, T., Becker, D.J. et al. Beowulf: A Parallel Workstation For Scientific Computation (1995) *Proceedings of the 24th International Conference on Parallel Processing*

[25] Kargoth, G., Lange, D.B. and Oshima, M. A Security Model For Aglets, *IEEE Internet Computing* Jul/Aug 1997

[26] Holland, J. H. *Hidden Order*, Perseus Books, 1995

[27] Langley, B., Paolucci, M., and Sycara, K., "Discovery of Infrastructure in Multi-Agent Systems. In *Agents Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001

[28] van Lengen R. H., Bähr J. T. "Visualisation and Debugging of Decentralised Information Ecosystems" To appear in Proc. of the Dagstuhl Seminar 01211 "Software Visualization", Lecture Notes in ComputerScience, Springer-Verlag, Dagstuhl, Germany, September, 2001.

[29] Nowostawski M., Bush G., Purvis M., Cranefield S. "A Multi-Level Approach and Infrastructure for Agent-Oriented Software Development" Proc. 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 5th International Conference on Autonomous Agents, Montreal 2001.