Self-Organising Applications Using Lightweight Agents

Paul Marrow[1] Manolis Koubarakis[2]

[1]Pervasive ICT Research Centre, BT plc, Orion 1 PP 12, Adastral Park, Martlesham Heath, Ipswich IP5 3RE, United Kingdom
paul.marrow@bt.com
[2]Intelligent Systems Laboratory, Dept. of Electronic and Computer Engineering, Technical University of Crete, University Campus - Kounoupidiana, 73100 Chania, Crete, Greece
manolis@intelligence.tuc.gr

Abstract. Self-organisation in nature is responsible for many complex and persistent phenomena. This suggests that self-organisation may be useful in the creation of complex applications. Multiagent systems use multiple agents to execute complex activities, and thus may be a basis for self-organising applications. In this paper we describe applications using self-organisation based upon the DIET multi-agent platform that supports lightweight agents. Multi-agent systems can be created that support decentralisation, scalability and adaptability. We show that these application properties are useful for information sharing in mobile communities via self-organising among middle agents, and via peer-to-peer interaction between agents.

1. Introduction

The complexity of the living world [5, 8] and of physical systems [31] has attracted much interest as a source of inspiration for applications. The persistence and ubiquity of the natural world suggests that self-organisation may be a means of constructing complex applications [2, 14]. Self-organising systems [14] function without central control, and through local interactions. Multi-agent systems (such as Farm [17], JADE [21], JAF [41], GAIA [46] and ZEUS [32]) provide a means of programming systems distributed across many autonomous entities. This suggests that they may be a useful basis for implementing self-organising systems, providing they have appropriate properties, including adaptability, scalability and decentralisation.

Self-organising systems need to be able to control and adapt their properties in a robust manner. This means that they must have means of control of their individual components that are robust to individual component change or failure. They also need to adapt to changing demands placed on them, as demands for resources change. And they need means of adapting performance in order to identify optimal performance.

Where systems consist of many agents questions arise of how to control their activities. The client-server model has been widely used in an agent context, as

elsewhere in computing. Middle agents (sometimes called brokers in certain types of interaction [11]) can be introduced as intermediaries between different classes of agents [11, 23, 44]. Although often extremely productive, both these configurations raise problems of the reliability of key elements if the failure of key components is possible: of the server or the middle-agents. Alternatively fully decentralised applications can use a pure peer-to-peer configuration [33], where each node is equal. Agent systems have been used to construct peer-to-peer architectures [2], and have proved useful for load-balancing applications [30]. These systems suggest that decentralisation is a useful property to ensure robust self-organising applications.

Another property that may be useful for self-organised systems is that of scalability. Scalability has been referred to as the capability to adapt a required performance level or number of users simply by adding resources to a system [38]. This might refer to changes in hardware as well as software, but here because of our focus on software agents we consider only changes in software. Discussion of scalability in multi-agent systems has considered various aspects of multi-agent performance [24, 35]. Changes in software resources as level of performance changes is clearly important, but organised change is also significant. Scalable changes in software resources might imply not more than linear change as system requirements vary. Ideally a scalable system should need less than linear increase in resources for higher performance levels. Self-organisation may make this easier, by giving some capability to anticipate changes in the environment, and adaptability will increase the capability to respond to changes in the environment. However, since self-organising applications may not be able to predict completely the level of demands upon them, some capability for scalability will be useful.

Decentralised control combined with scalability can produce effective performance. An additional property of self-organised applications is to be able to adapt to changing environmental conditions, including changing requirements of users. Individual components need to be able to change their behaviour in response to external stimuli, to be able to adapt. Adaptation in multi-agent systems may take place by a variety of different means [12, 29, 37]. Because self-organising applications depend on local interaction between components, adaptation will arise in part from interaction between individual components [e.g. 12, 27]. A self-organising system that has some capability for adaptability when combined with decentralisation and scalability should be able to support robust applications.

This paper presents some examples of applications that use self-organisation to achieve their results. In the next section we outline the system used, based on a general-purpose multi-agent platform, the DIET Agents platform [13, 16, 28]. The architecture of the agent platform is described, along with key components and properties that support self-organised applications. In section three two applications that use self-organised agent systems are described: the self-organising communities application and the P2P-DIET peer-to-peer service. Finally we consider how these applications show properties consistent with robust self-organised systems.

2. System Outline

2.1 The DIET Agents Platform

The applications described in this paper have been implemented using the DIET Agents platform [13, 16, 28]. This software platform draws inspiration from nature, where many living organisms interact in diverse ways so as produce complex ecosystems [43]. While some animals do exhibit a degree of intelligence in dealing with their environment, many organisms with little or no individual intelligence can deal productively with environmental challenges through group behaviour (e.g. [6]). Consequently the platform is designed around agents with minimal properties, with limited individual capabilities. DIET agents are not assumed to be highly intelligent or to use complex communication protocols. Intelligent behaviour can emerge from the interaction between agents.

The DIET Agents platform is designed in a three-layer architecture (Figure 1). The lowest layer, the core layer, contains the DIET kernel, enabling DIET environments, providing the basic capabilities for agent creation. The kernel also provides for connections between agents, which allows messages to be passed between them. The core layer also contains basic support for debugging and visualisation.

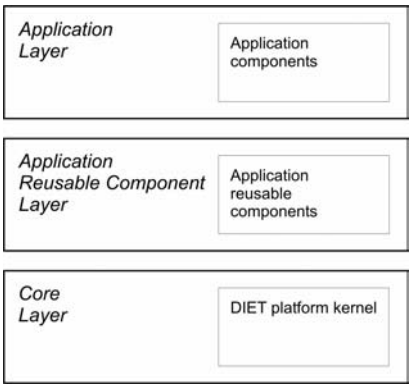

Fig. 1. DIET three-layer architecture

The application reusable component layer contains software components that are reusable between multiple applications, but are not essential for inclusion in the core layer. Examples of the components included are ones supporting remote communication and event scheduling.

The application layer is the top layer. It contains code specific to particular applications, as well as debugging and visualisation code that may be application specific. Establishing agents in this way enables the multi-agent system so created to

be linked to a visualisation system so that users can visualise agent behaviour (Figure 2, [25]).

The architecture described here simplifies the development of applications, as core classes required for multi-agent systems are provided in a manner that can be easily extended without limiting too much the direction of potential applications. The DIET core platform has been released as Open Source under the GPL Open Source licence. This should enable a wider community of developers to use this system. More information is given on the DIET Open Source web site [13].

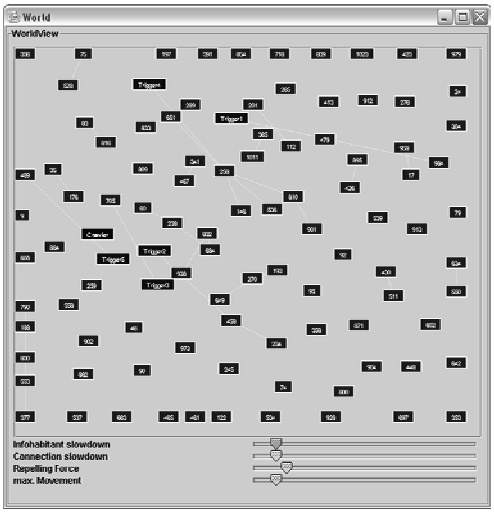

Fig. 2. DIET visualisation (icons indicate agents, sliders provide settings).

2.2 Kernel properties

The DIET kernel defines a hierarchy of elements that provide a basis for creation of multi-agent systems:

- Worlds
- Environments
- Agents
- Connections
- Messages.

Autonomously executing agents require environments in which to reside. Environments implement a "physics" for DIET agents, in which agents can be created or destroyed, migrate or communicate. Worlds are placeholders for environments - a world manages functionality shared between environments. A typical application may only create one world, so that there is only one world per Java Virtual Machine. However if an agent application is running in a browser, where there will be multiple

applets in one JVM, each applet will have a world and there will be multiple worlds in one JVM. A world is also the access point to the kernel for debugging and visualisation components.

Agents are at the centre of the DIET element hierarchy, but each agent has very limited initial properties in comparison with some other agent systems where focus is more on single complex agents. Here, where we are interested more in the properties of groups of agents, individual agents are reactive and lightweight in terms of their individual requirements of computational resources, and consequently have limited individual capabilities.

Individual agents are defined in terms of addresses. This consists of an *environment address*, determined by the environment in which it resides, and an *agent identity*, defined when the agent is created. The agent identity has two components, a *name tag* specific to that agent, and a *family tag* common to other agents with the same functionality. The family tag allows for groups of agents with common function to be identified by an application. The name tag allows individual agents to be differentiated.

Agents have very limited initial behaviour (which can be extended as required by application developers). The environment provides functionality for agents to *create* other agents, *migrate* to different environments, *communicate* with other agents, and to *self-destruct* if they are no longer required (but they cannot destroy other agents). Communication between agents is local, taking place within environments. The agents form a connection before they can communicate, allowing both agents to operate as one group. Which agents form connections and then communication locally is defined as part of an application.

Agents can migrate between environments once they know the address of the destination environment. It is possible to distinguish between *neighbouring environments* and others further away, but agents can migrate to all of them. Communication between agents in different environments is possible, but must take place indirectly via a link between different environments. This can be formed via a *Carrier Pigeon* agent, provided in the Application Reusable Component layer.

The kernel implementation for agent behaviour is "resource constrained" and "fail-fast". The kernel actions are *resource constrained* because there are explicit limits on the resources that can be used. The buffer where each agent receives incoming messages is of limited size. The number of threads that DIET agents are able to use in a DIET world is defined by a parameter specified by the user when the world is created. The kernel actions are *fail-fast* because when an action cannot be executed instantaneously, it fails immediately. The kernel does not retry the actions later and it does not block or guard execution until it has successfully executed the action. So when an attempt is made to create a new agent, but there is no thread available for it, this will fail.

The fail-fast, resource constrained implementation of the kernel actions protects the system against overload. When an agent attempts to send a message to an agent whose message buffer is already full, the message is rejected. When the system becomes overloaded, it offers basic protection by rejecting actions and throwing exceptions, which provide feedback to agents. The agents then have an opportunity to change their behaviour, rather than continuing their original processes.

2.3 Application properties

The kernel properties of the DIET system described above give some advantages in terms of application properties.

2.3.1 Decentralisation

Applications constructed on the DIET platform are based around agents in environments. The minimum configuration is a single environment, with an arbitrary number of agents included, but agents can be distributed across multiple environments and multiple worlds if needed. No agent need be the controlling agent, or server, so there is no obligation for a client-server architecture. Consequently the DIET platform is appropriate for peer-to-peer applications [6] which can be used to implement decentralised control. The flexibility of peer-to-peer configurations [33] means that decentralisation does not prevent some sort of localised centralisation, where this is useful [27].

2.3.2 Scalability

Low resource use by agents in a multi-agent system can facilitate scalability in terms of adapting performance through adding resources. In the DIET platform, low resource demand by individual agents, combined with thread-sharing between agents, means that a very large number of agents can be maintained simultaneously. The specific amount of resources required by each agent, and the resource overhead of the platform are application-dependent, but the initial system design facilitates low resource requirements. DIET agents are designed such that each is only using one single thread (at most). Thread-sharing ensures that agents do not need more threads than are available. If insufficient threads are available, agents can temporarily give up their thread when they do not need it, only to reclaim it when needed again. This means that one thread can be shared between a number of agents.

Because DIET agents can migrate between environments, and environments can be created on multiple machines, it is possible to connect machines to support many agents. A 32-processor Beowulf cluster has been used to support 100000 agents [26]. There is no reason why this should be an ultimate limit, because thread-sharing can be exploited to give even greater scalability. If there is a reason to develop an application needing even more agents, this is possible.

2.3.3 Adaptability

Individual agents in multi-agent systems need to be able to change their behaviour in response to demands from their environment, that is to adapt their behaviour. This capability to adapt individual behaviour may be produced through defining alternative behaviours at an individual level, but this may be inefficient in terms of agent software design and requires some anticipation of future performance. In the DIET Agents platform this is not a possible approach as DIET agents are initially created

with only very limited capabilities. So DIET agents have to be able to adapt by collaboration among a large number of agents. Although individual agents may only have characteristics allowing a few behaviours, alternative agents can be selected to adapt an overall population to new behaviours.

A demonstration of how this can work comes in the use of DIET agents as individuals in an evolutionary algorithm, using such an algorithm to evolve agent preferences [16, 27]. Agent preferences are used to respond to user preferences. The application is a collaboration tool, where each user interacts with the application via a user agent, which is deployed in a DIET environment. The user agent carries information about its user's interests, indicating the area of collaboration the user would like to engage in. Multiple users participate via this application, either on the same or different machines, each being represented by a user agent in its own DIET environment. Different DIET environments may be on the same or different machines, depending upon the location of the users. Environments are linked in a peer network, intended to approximate a fully connected decentralised peer network [27]. User agents each remain in a single environment but deploy scout agents to interact via the peer network. The preference of scout agents for environments is selected via an evolutionary algorithm using scout agents as individuals. The scout agent genomes that will have most success are those that will direct scout agents to environments that allow them to meet other agents that represent users with similar interests. Over multiple generations populations of scout agents will adapt their behaviour to facilitate interaction between distributed users [27].

This algorithm represents a means of mediating collaboration between distributed users. Although individual user and scout agents do not have much capability to adapt their behaviour individually, by selection in a population considerable adaptability can be achieved in support of collaborative behaviour.

3. Self-Organised Applications

Because the DIET platform emphasises lightweight agents with limited initial capabilities, complex applications are unlikely to arise from one or a few agents. Interactions between agents are of considerable importance in establishing a diversity of agent behaviour, and thus in supporting self-organisation. The DIET kernel provides simple interactions (e.g. communication or connection), but these can be extended if other forms of interaction are needed. In order to reduce the need for agent extension, middle agents can be used, and these can result in complex behaviour through groups of agents. Groups of agents can interact into persistent communities that arise from the interactions between the agents rather than from some central, and so are self-organised. The two examples here deal with information sharing in two different application contexts: self-organising communities [42] and P2P-DIET [22].

3.1 Self-organising communities

The construction of a flexible organisation for information sharing among multiple users can be done in several ways. Centralised solutions such as SHADE [23] or InfoSleuth [4] can provide effective solutions, but they do depend on a single middle agent not getting overloaded with information. Decentralised solutions (such as the virtual integrated distributed database facilitator [39] and Abrose [1]) have an advantage in this context, and can self-organise to respond to changes in user behaviour or requirements.

The self-organising communities application [42] facilitates community formation among users with common interests within a larger community made up of multiple sets of interest. Personal user agents each represent users. User agents first register with a middle agent. Each user agent can go on to register with more than one, but they do not have to. User agents send queries to middle agents, about the sort of information they are looking for. Given these queries, each middle agent then searches among the information it holds from the other user agents that have registered with it. If it can respond to a query based on this information then the search is rapidly completed. Otherwise, the middle agent can communicate with other middle agents in order to try to obtain the information from them. Once the communication with other middle agents has been carried out, the middle agent relays results of the search to the user agent. Having done this, the middle agent examines whether the search was successful for the user agent, in that there was an information provider matching the information query. If so, the user agents of the user and provider can gain positive awards. If not, the user agent of the information requestor gets a negative mark to represent the extra load it placed on the system.

Following the assignment of rewards after a search, the middle agent concerned checks whether both the requestor and provider user agents are within its local group. If not, the middle agent of the requestor then transfers the provider user agent to its group, so that both user agents are in the same group. User agents with lower awards move towards user agents with higher awards. The consequence of the award scheme following queries is that the processing of queries from users stimulates the formation of user communities, made up of users with common interests. The querying behaviour of user agents in effect builds up a profile of their interests, and this profile is updated by successive queries. Because this uses the DIET platform, it is a highly scalable process, that continues to operate highly efficiently even as very large numbers of users are involved [42, 10].

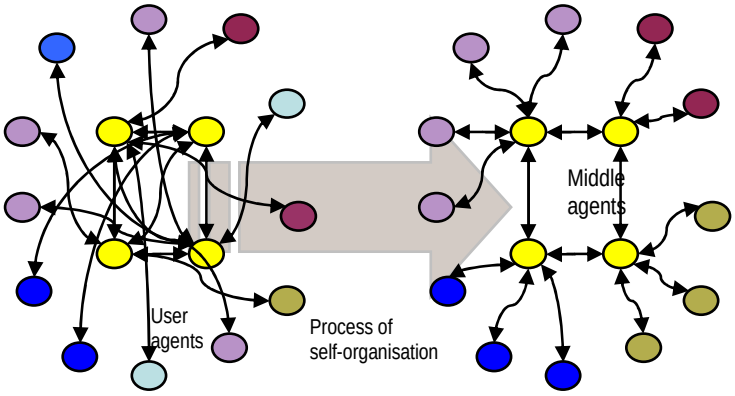

Fig. 3. Self organisation of agent communities - different colours represent different user interests.

Experiments were carried out with simulated data on the formation of user communities, and time and efficiency of searches based on the user communities formed around middle agents [42]. Multiple experiments always showed self-organisation in terms of the formation of user communities made up of user agents around middle agents. In some cases one middle agent was responsible for one community of users, in other cases for several. However some middle agents lost all their user agent connections during the process of community formation. This was not a failure of the process of self-organisation, rather a reflection of the matching of a varying number of user agents and middle agents, with efficient selection of both user agents by middle agents and implicitly, middle agents by user agents.
Measurement of self-organisation activity could be carried out through analysis of the success rate of queries, since each query depended upon organisation of user agents into a community around a middle agent to get a successful response from another user agent. Investigation of success rates of query results and time taken to search data showed an increase in success rate once communities of users had started to form, culminating in a success rate close to the optimal rate once user communities had reached a stable state. The query time for individual queries also declined, as well as the success rate increasing, as a consequence of the formation of user communities around middle agents. The number of queries per user taken for communities to reach a stable state increased with the number of users, but in a near linear manner, so that performance scaled effectively [42].

3.2 Implementing Peer-to-Peer Systems with P2P-DIET

The DIET platform has also been used to implement the extensible P2P service P2P-DIET [19, 20, 22, 34]. P2P-DIET is a super-peer system and has two kinds of nodes: super-peers and clients (see Fig. 4). Super-peers are equal and have the same responsibilities. Each super-peer serves a fraction of the clients and keeps indices on the resources of those clients to be able to answer queries efficiently. Clients can run on user computers where resources are also stored. Clients interact directly with one another to access resources.

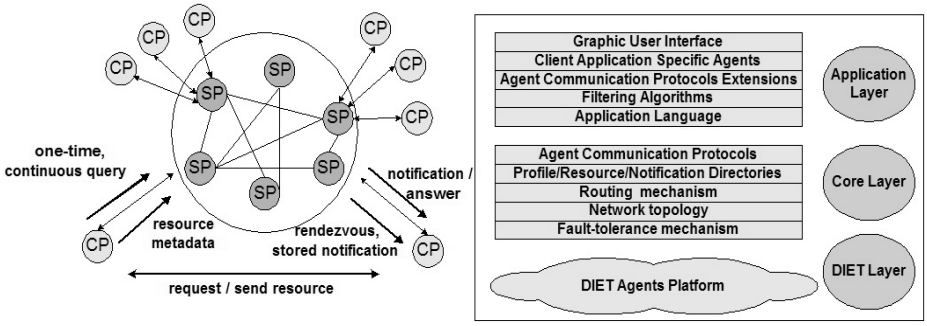

Fig. 4. P2P DIET

P2P-DIET supports the typical one-time query scenario of P2P networks. A client can send a query (e.g., "I want music by Moby") to its access point and the access point will broadcast this query to all super-peers. In this way, answers will be produced for all matching network resources. Answers are returned to the access point of the client originating the query and are then passed to the client for further processing. P2P-DIET also supports long-standing (continuous) query scenarios (e.g., "Notify me when a song of Moby becomes available"). Clients may subscribe to the system with a continuous query expressing their long-standing information needs. Whenever a resource is published at an access point, P2P-DIET makes sure that clients with profiles matching the metadata of this resource are notified.

A client can be connected to P2P-DIET through a single super-peer node, which is the access point of the client. Clients are allowed to migrate to a different access point and can use dynamic IP addresses. Clients can connect, disconnect or even leave the system silently at any time. When a client is off-line, notifications matching its continuous queries are stored by the access point of the client and are delivered to it the next time that it connects to the network. A similar situation is when a client A requests a resource, but the resource owner client B is not on-line. In this case, the client A may request a rendezvous with the resource. When client B later-on reconnects to the network, its access point informs it that the resource must be delivered to the access point of the client. P2P-DIET provides message authentication and message encryption using public key cryptography. Public/private keys are also

used to securely identify peers since it is not possible to identify a peer from its IP address because peers may use dynamic IP addresses.

Nodes in P2P-DIET are implemented as DIET environments where different types of agents live. The P2P-DIET implementation makes use of the capabilities of lightweight mobile agents offered by the DIET kernel to implement various local management tasks and P2P protocols. For example, in each super-peer environment, a *data management* agent keeps indices on resource meta-data and continuous queries to achieve scalable query processing and filtering by each super-peer. In each super-peer environment, a *router agent* achieves correct flow of network messages by using shortest paths and minimum-weight spanning trees. For each ad-hoc query posed by a user, a *query answering* agent starts from the user's client environment and, using information from router agents, migrates to all super-peers to find all resources that match the user query. Similarly, for each continuous query subscribed by a user, a s*ubscriber agent* starts from the user's client environment and migrates to appropriate super-peer environments to subscribe the query. Migration is performed towards all super-peers that might end-up with resource metadata matching the query and is only constrained by taking into account query subsumption relations. Subscription migration has the effect that filtering takes place closer to the clients posting the resources to curtail message propagation in the network.

P2P-DIET was built with the intention to show that the DIET Agents platform with its minimal agents and self-organization capabilities can be used to develop large scale P2P applications. The P2P-DIET application takes advantage of self-organisation at two levels: the super-peer network and the DIET Agents platform that is used to realize P2P-DIET. At the first level, P2P-DIET super-peers self-organize into a network which deals efficiently with pull and push information requests while, at the same time, adapting to super-peer joins, leaves or failures. At the second level, lightweight DIET Agents self-organize to implement P2P-DIET functionalities. For example, query answering agents begin from a client node and migrate to remote nodes with the purpose of answering a one-time query. At each node that they arrive, they interact with router agents and data management agents to decide their next destination node or to compute partial query answers. The next operation to be performed by an agent is not determined by any kind of higher level of control. On the contrary, its current status and the state of its environment determine agent decisions. For example, an agent might choose to replicate itself if there are more than one possible route to follow for answering a query. Newly created agents are totally independent to travel around the network and collect query answers.

P2P-DIET is an extensible system for the development of applications in need of the two scenarios discussed above (see the layered view of the system on the right-hand side of Figure 4). In the current demo of the system publications and subscriptions are expressed using a well-understood attribute-value model called *AWPS* in [22]. *AWPS* is based on named attributes with value text interpreted under the Boolean and VSM or LSI models. The query language of *AWPS* allows Boolean combinations of comparisons *A op v*, where *A* is an attribute, *v* is a text value and *op* is one of the operators ``equals", ``contains" or ``similar" (``equals" and ``contains" are Boolean operators and ``similar" is interpreted using the VSM or LSI model of Information Retrieval. [New] provides detailed performance analysis of some of the indexing algorithms for *AWPS* used in P2P-DIET and demonstrates very good

efficiency and scalability properties (e.g., upon receiving a new publication, a P2P-DIET node can filter 3 millions of long-standing queries in just under 200 milliseconds).

The readers might also find interesting the recent extension of P2P-DIET with RDF-based languages for metadata and queries and its integration with the super-peer network EDUTELLA [9].

4. Discussion

We have considered two examples of self-organised application that build upon the properties of the DIET Agents platform. The self-organising communities application demonstrates the capability to organise user agents into communities of interest that facilitate information exchange. This community formation varies with increasing numbers of users and hence increasing numbers of agents, but produces a mean time of response to query that scales almost linearly with number of users. This shows not only a versatility for different sizes of problem spaces with different numbers of users, but also, since community formation appears to always occur, albeit at slightly different rates, an important feature of self-organisation that appears not to be disrupted by different starting conditions. In this case self-organisation between agents leads to conditions that are more productive for individual agents than those in which they started. Furthermore, it ensures reliability of the application as queries from user agents are solved.

P2P-DIET was built with the intention to show that the DIET Agents platform with its minimal agents and self-organization capabilities can be used to develop large scale P2P applications. Although distributed P2P applications can be built without using agents, the DIET agents platform allowed us to develop such applications quickly and effectively. However, more experience with such applications is necessary in order to come up with a methodology for the development of software systems using lightweight agents (presumably, this methodology will be based on traditional software engineering principles and new inspirations from self-organization). The use of self-organization and the sophisticated indexing schemes of [40] helped us to achieve high performance and scalability in P2P-DIET. We have not carried out yet detailed experimentations of P2P-DIET beyond these reported in [40], so it is not clear at this stage how much of the scalability of P2P-DIET is due to indexing and how much to the self-organizing protocols utilized.

Both the examples discussed here address problems of information retrieval among a diverse community of users. Information retrieval (e.g. [3]) does not necessarily require either multi-agent systems or self-organising properties. Multi-agent based applications bring the capability to partition information across multiple agents, and thus to break up the problem into smaller parts. This also allows for decentralisation of the management of the application, giving greater robustness. An element of self-organisation makes control easier even in a decentralised network, as appropriate elements associate without additional external stimulus.

Both applications implement peer-to-peer networks in the form of super peer networks. These have been shown to have advantages over pure peer-to-peer networks, in terms of efficiency of search [45], as well as over client-server architectures. The applications discussed here show some of the usefulness of super peer networks. Further work needs to be done to investigate the performance of applications based on these networks in relation to other peer-to-peer information retrieval applications (e.g. [36]), and the consequences of adjusting properties of the networks used on application performance.

These examples have shown ways how lightweight multi-agent systems can be used to leverage self-organisation among agents to produce information-managing applications. The applications here draw upon relatively simple forms of self-organisation and of information representation. Next steps for the development of such applications include considering extensions to the network architecture linking agents, and investigating different types of queries or responses from such agent-based collaboration tools. The DIET Agents platform suggests how to draw inspiration from natural systems in developing multi-agent systems that are scalable, flexible and decentralised, while retaining very light-weight agents that make minimal demands on computational resources. Further development of applications in this area must follow further the potential for peer-to-peer computing [2, 15, 33] and autonomic computing [18] to support the development of robust applications. The interest in peer-to-peer technologies in combination with agent systems, suggests an important role for agent systems such as those of DIET Agents as the basis for self-organised applications.

Acknowledgements

We would like to thank the participants of the DIET project for their contributions, and the IST/FET initiative of the European Commission for making the DIET project possible.

References

[1] Athanassiou, E Chirichenescu, D Gleizes, MP Glize, P Lakoumentas, N Sclenker, H Leger, A Moreno, JI 1999 Abrose: A cooperative multi-agent based framework for marketplace. IATA, Stockholm, 1999.

[2] Babaoglu, O Meling, H Montresor, A 2002 Anthill: A framework for the development of ant-based peer-to-peer systems. Proc. IEEE Intl. Conf. Distributed Computer Systems, 2002, pages 15-22.

[3] Baeza-Yates, R Ribeiro-Neto B 1999 *Modern Information Retrieval*. Addison-Wesley.

[4] Bayardo, RJ Jr, Bohrer, W Brice, A Cichocki, J Fowler, A Helal, V Kashyap, T Ksiezyk, G Martin, M Nodine, M Rashid, M Rusinkiewicz, R Shea, C Unnikrishnan, A Unruh, A Woelk, D 1997 Infosleuth: agent-based semantic integration of information in open and dynamic environments. Proc. ACM SIGMOD-97, 1997.

[5] Bonner, JT 1988 *The Evolution of Complexity by means of Natural Selection.* Princeton University Press.

[6] Bonsma, E Hoile, C 2002 A distributed implementation of the SWAN peer-to-peer look-up system using mobile agents. In: 1st Workshop on Agents and Peer-to-Peer Computing, Bologna, 2002.
[7] Bourke, AFG Franks, N 1995 *Social Evolution in Ants.* Princeton University Press.
[8] Camazine, S, Deneuborg, J-L, Franks, NR, Sneyd, J Theraulaz, G Bonabeau, E 2001 *Self-Organization in Biological Systems.* Princeton University Press.
[9] Chirita, P-A Idreos, S Koubarakis, M Nejdl, W 2004 Publish/Subscribe for RDF-based P2P Networks. European Semantic Web Symposium 2004, LNCS 3053, pp. 182-197. Springer.
[10] Cid-Sueiro, J Wang, F 2002 A scalability analysis of self-organising agent communities. Learning02 Workshop, Madrid, 2002.
[11] Decker, K Sycara, K Williamson, M 1997 Middle-agents for the internet. Proc. IJCAI-97, pp. 578-583.
[12] De Wilde, P Chli, M Correira, L Ribeiro, R Mariano, P Abramov, V Goossenaerts, J 2003 Adapting populations of agents. In: *Adaptive Agents and Multi-Agent Systems*, Alonso, E., Kudenko, D. & Kazakov, D. (eds.) pp. 110-124, LNAI 2636, Springer.
[13] DIET Open Source web site: http://diet-agents.sourceforge.net/Index.html
[14] Di Marzo Serugendo, G Foukia, N Hassas, S Karageorgos, A Mostefaoui, SK Rana, OF Ulieru, M Valckenaers, P Van Aart, C 2004 Self-organisation: paradigms and applications. In: *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, Di Marzo Serugendo, G Karageorgos, A Rana, OF Zambonelli, F (eds.), pp. 1-19. LNA 2977, Springer.
[15] El-Ansary, S Aurell, E Brand P Haridi, S 2004 A Physics Inspired performance evaluation of a structured peer-to-peer overlay network. Proc. Workshop on Self-* Properties in Complex Information Systems, 2004, Bertinoro, Italy.
[16] Hoile, C Wang, F Bonsma, E Marrow, P 2002 Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System. Proc. 1st Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS2002), 2002, pp. 623-630.
[17] Horling, B Mailer, R Lesser, V 2004 Farm: a scalable environment for multi-agent development and evaluation. In: *Advances in Software Engineering for Multi-Agent Systems*, Lucena, C Garcia, A Romanovsky, A Castro, J Alencar, P (eds.) pp. 220-237. Springer.
[18] Horn P 2001 Autonomic computing: IBM's perspective on the state of information technology. IBM Autonomic computing manifesto, IBM 2001.
[19] Idreos, S Koubarakis, M Tryfonopoulos, C 2004 P2P-DIET: An Extensible P2P Service that Unifies Ad-hoc and Continuous Querying in Super-Peer Networks. ACM SIGMOD Conference 2004, Demo Paper, pp. 933-934.
[20] Idreos, S Tryfonopoulos, C Koubarakis, M Drougas, Y 2004 Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: The P2P-DIET Approach. EDBT 2004 Workshop on Peer-to-Peer Computing and Databases, LNCS 3268, Springer. pp. 496-505.
[21] JADE web site: http://jade.tilab.com/
[22] Koubarakis, M Tryfonopoulos, C Idreos, S Drougas, Y 2003 Selective Information Dissemination in P2P Networks: Problems and Solutions. *ACM SIGMOD Record*, Special issue on Peer-to-Peer Data Management, Aberer, K (ed.), 32(3), September 2003.
[23] Kuokka, D Harada, L 1995 Supporting Information Retrieval via Matchmaking. In: AAAI Spring Symposium on Information Gathering, 1995.
[24] Lee, LC Nwana, HS Ndumu, DT De Wilde, P 1998 The stability, scalability and performance of multi-agent systems. BT Tech. J. 16(3), 94-102.
[25] Lengen, RH van, Bähr, T Hagen, H Marrow, P Bonsma, E Hoile, C 2004 Component based visualisation of DIET applications, *Proc. Dagstuhl Workshop on Scientific Visualisation 2003*, G.-P. Bonneau, T. Ertl and G.M. Nielson (eds.), Springer, 2004.

[26] Marrow, P 2001 Scalability in multi-agent systems: the DIET project. Workshop on Infrastructure for Agents, Multi-Agent Systems and Scalable Multi-Agent Systems at Autonomous Agents 2001.
[27] Marrow, P, Hoile, C Wang, F Bonsma, E 2003 Evolving preferences among emergent groups of agents. In: *Adaptive Agents and Multi-Agent Systems*, Alonso, E., Kudenko, D. & Kazakov, D. (eds.) LNAI 2636, Springer, 2003, pp. 159-173.
[28] Marrow, P Bonsma, E Wang, F Hoile, C 2003 DIET – a scalable, robust and adaptable multi-agent platform for information management. BT Tech. J. 21(4), 130-137.
[29] Menezes, R Tolksdorf, R 2004 Adaptiveness in Linda-Based Coordination Models. In: *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, Di Marzo Serugendo, G Karageorgos, A Rana, OF Zambonelli, F (eds.), pp. 212-213. LNAI 2977, Springer.
[30] Montresor, A Meling, H & Babaoglu, O. 2002 Messor: load-balancing through a swarm of autonomous agents. Proc. 1st Workshop on Agents and Peer-to-Peer Systems, Bologna, 2002.
[31] Nicholis, G Prigogine, I 1989 *Exploring Complexity: an introduction*. W.H. Freeman.
[32] Nwana, H Ndumu, D Lee, L Collis, J 1999 ZEUS: A toolkit for building distributed multiagent systems. Applied Artificial Intelligence J. 13(1), 129-186, 1999.
[33] Oram, A (ed.) 2001 *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly.
[34] P2P-DIET web site: http://www.intelligence.tuc.gr/p2pdiet
[35] Rana, OF Stout, K 2000 What is scalability in multiagent systems?. Proc. 4th Intl. Conf. on Autonomous Agents, Barcelona.
[36] Sartiani, C Manghi, P Ghelli, G Conforti, G 2004 XPeer: A self-organising XML P2P Database system. In: Proceedings of P2P&DB Workshop, EDBT Conference 2004, Heraklion, Crete, Greece.
[37] Splunter, S van Wijngaards, NJE Brazier, FMT 2003 Structuring agents for adaptation. In: *Adaptive Agents and Multi-Agent Systems*, Alonso, E., Kudenko, D. & Kazakov, D. (eds.) LNAI 2636, pp. 174-186. Springer.
[38] Steen, M van, S van der Zijden, and H. Sips 1998 Software engineering for scalable distributed applications. In Proceedings 22nd International Computer Software and Applications Conference (CompSac), 1998.
[39] Takada, Y Mohri, T Fujii, H 1998 Multi-agent system for virtually integrating distributed databases. Fujitsu Sci. Tech. J 34(2), 245-255.
[40] Tryfonopoulos, C Koubarakis, M Drougas, Y 2004 Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators, Proceedings of the 27th Annual ACM SIGIR Conference. July 25-July 29, 2004, Sheffield, U
[41] Vincent, R Horling, B Lesser, V 2001 An agent infrastructure to build and evaluate multi-agent systems: the Java Agent Framework and Multi-Agent System Simulator. In: *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Wagner & Rana (eds.). LNAI 1887, Springer.
[42] Wang, F 2002 Self-organising communities formed by middle agents. Proc. 1st Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS2002), 2002, pp. 1333-1339.
[43] Waring, RH 1988 Ecosystems: fluxes of matter and energy. In: Cherret, JM (ed.), *Ecological Concepts*, pp. 17-42, Blackwell Scientific.
[44] Wong, HC Sycara, K 2000 A taxonomy of middle-agents for the Internet. Proc. 4th Intl. Conf. Multi-Agent Systems.
[45] Yang, B Garcia-Molina H 2003 Designing a super-peer network. IEEE International Conference on Data Engineering, 2003.
[46] Zambonelli, F Jennnings, NR Wooldridge, M 2003 Developing multiagent systems: the GAIA methodology. ACM Transactions on Software Engineering and Methodology 12, 317-370.