Whitestein Series in Software Agent Technologies, 1–13

Nexus: Self-organising Agent-based Peer-to-Peer Middleware for Battlespace Support

Alex Healing, Robert Ghanea-Hercock, Hakan Duman and Michal Jakob

Abstract. The problem facing the security and defence communities is the volume, complexity and timeliness of information. In particular the ability to locate and access the right ICT service at the right time is crucial to achieving real-time responsiveness and situational awareness. The Nexus system is a Peer-to-Peer (P2P) agent-based middleware that creates a fully distributed and highly resilient Service Oriented Architecture (SOA). The combination of a structured P2P overlay network and autonomous service discovery, delivers a powerful capability to support real-time operations in either security or defence applications. This paper outlines the overall architecture of the Nexus system and its application in a defence scenario with a detailed review of the service selection algorithm utilised, termed Mercury. Mercury provides an autonomous, efficient and distributed service selection framework and collaborative algorithms for SOA construction and real-time adaptation.

1. Introduction

Future military force requirements will demand a migration towards ever increasing levels of ICT automation and self-organising capability. This is a simple function of reduced administrative support, increasingly complex networked systems and the ever shrinking time available for response. In addition the need for shared situational awareness across tactical and coalition spheres makes manual service configuration a logistic nightmare. This chapter reviews a solution based on combing the best features of P2P and SOA approaches to create a self-* service platform. An overview of the Nexus autonomic middleware is first given, followed by an in-depth technical discussion of one of its components - adaptive service selection.

2. Approach

The first phase of the Nexus project [7, 9] demonstrated the value of an agent-based P2P middleware for the discovery and fusion of NEC services. The Nexus middleware is based on three key paradigms: P2P computing, autonomous agents and SOA [5]; all of which have been identified as key components of future NEC network architectures [1]. Existing implementations of SOA, as applied in the civil domain, suffer from several issues that make them unsuitable for volatile environments. These include centralized service discovery and process orchestration, and fixed manually specified workflows. These factors lead to fragile, non-adaptive and difficult-to-maintain network applications. The aim is to develop a hardened, agent-based SOA implementation that meets the strict reliability requirements of the NEC domain and accommodates the needs of network-centric information fusion applications. More specifically, the following capabilities are being developed either as a direct part of Nexus II middleware or by integrating technologies from other projects within the Hyperion cluster [2]:

- Seamless and reliable service delivery in volatile environments
- Request prioritisation and load-balancing
- Resilience to volatility of the underlying network infrastructure: by adopting a peer-to-peer architecture Nexus maintains its operability even if a large subset of services or the network itself becomes unavailable.
- Decentralised service discovery whereby networked resources are discovered based on their advertised properties and real-time information regarding their dynamic attributes without reliance on a centralised repository.
- Semantic and adaptive service selection based on dynamically maintained quality-of-service profiles.
- Proactive monitoring and automated service substitution: The state of services is actively monitored and should a failure occur the failed resource is rapidly substituted with the closest alternative, preserving the overall capability.
- Filtering of information services based on their semantic relevance to the user as well as imposing some structure at the messaging layer of the middleware allowing bandwidth to be conserved.

In order to offer the necessary resilience Nexus adopts an entirely decentralised approach. At the lowest level, a P2P overlay network is constructed, either directly or indirectly, connecting each of the nodes in the network running Nexus with each other. Similar to [12, 18], the overlay network is then coupled with component-model technologies which in our case offer a Publish/Subscribe (Pub/Sub) structured messaging layer from which higher level management of the network can be constructed.

Each Nexus node can host a number of services and these are made available through the middleware by means of advertising their associated metadata on the messaging layer. Users of Nexus are required to connect to only a single node from where the middleware allows them to discover resources throughout the network

and manage their view and usage of the information services according to their requirements.

We adopt the Autonomic Computing [10, 13] paradigm which introduces self-* capabilities to allow Nexus to intelligently and autonomously handle the dynamic environment for which it is intended; including changing requirements of users, unreliable service availability, or failure of the underlying physical network.

Nexus is entirely implemented in the Java programming language and relies on several open-source third party libraries. In particular, the current embodiment of Nexus builds on an open-source P2P implementation of Java Message Service (JMS) [8] to provide the majority of the functionality of the bottom two layers in Fig. 1.

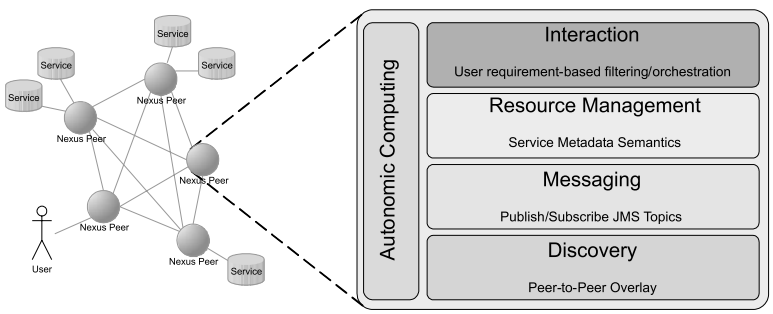

FIGURE 1. Nexus Autonomic Middleware Architecture.

IP multicast is used to discover other Nexus peers and construct the overlay whereby each peer advertises itself on a common channel thus allowing each peer to know the presence of others. JMS topics provide Pub/Sub functionality for the message-oriented component of Nexus and allow for information service advertisements to be structured in their transmission across the network. Each peer acts as a message broker and routes messages to peers that are subscribed to the topic on which the message was published. The topics can be structured into a hierarchy, allowing one to subscribe to only messages concerning a specific subset of services. To some degree, the semantics relating to the service descriptions in the resource layer can be exposed to the messaging structure in the layer below. The routing of messages throughout the overlay can therefore be linked to the semantic relevance of the resources that the messages describe to each peer. This capability is of value in reducing the overall bandwidth requirements of the supported applications and services.

There are numerous aspects of the architecture to which autonomic computing principles may be applied. For example, the driving of the aforementioned messaging structure by the service metadata may be an autonomous process. At

the lowest level, the overlay network is self-organising in that changes to the topology are dealt with seamlessly allowing for new peers joining the network to be discovered by others as well as the overlay to adapt their routing when peers are removed from the network.

The focus of the autonomic capability, however, is at the upper levels of the system model. Agent-based approaches to service orchestration are being investigated as well as methods to enable self-healing to fulfil a given user service requirement in the case of a service failure. These two aspects are related and both rely on the system understanding, to some degree, (a) what the user requirements are, (b) what services are available and how they relate, (c) the expected Quality-of-Service (QoS) services can deliver in a certain context.

Service selection is a key element of a resilient service-oriented middleware providing means to route service requests to the providers which best fit for the task. The problem becomes increasingly difficult in volatile environments where the availability and performance of service providers can change rapidly. In such situations, it is essential that the middleware has the ability to keep track of constant changes and updates its selection procedures accordingly. Decentralised adaptive mechanisms are a promising way in which this can be achieved. In the following section, we describe Mercury adaptive service selection which has been developed as part of the Nexus middleware. The description serves as an example of a concrete implementation of some of the autonomic principles mentioned previously.

3. Autonomic Computing Case Study - Mercury Adaptive Service Selection

3.1. Overview

The Mercury framework [6] is designed for application within an SOA and as such assumes a network of interconnected devices, each capable of hosting a number of processes. The processes may adopt at least one of two roles: service provider or consumer. Service providers offer capabilities that other devices (consumers) can access and use. Mercury-based service-selection takes place on the consumer side and assumes that for every device where there is a service consumer, a selector agent is hosted. Thus in Nexus, we envisage embedding a selector agent at each Nexus node.

Mercury relies on there being some service discovery mechanism in the SOA in order to gain a list of functionally capable service providers for a particular task. This functional discovery is based on those attributes that the service providers advertise in their description and can be provided by other components of Nexus. The Mercury selector agents then use the list of capable services as a basis for further finer-grained, non-functional selection. This is achieved by aggregating QoS data for each of the providers through the consumer's experience of them and ranking them accordingly. The result is a model of selection learnt over time which

distinguishes those services which are best at performing the task in terms of the QoS they are expected to deliver.

The QoS data of providers is stored in an instance-based model local to each selector agent and is parameterised by the task, as well as the context. Context is defined as the set of attributes which are external to the task requirements but nevertheless may influence the performance of providers (e.g., performing differently at different times of day). A particular service selector therefore builds up a model of how suited each provider is at fulfilling each particular task in each context.

The main contribution is the design of an efficient distributed service selection framework and (collaborative) algorithms for its construction and real-time adaptation. The learning techniques used are similar to those in reinforcement learning [16], however are novel in the degree to which they are adaptive. Specifically, a decision function is employed (Fig. 2) to ensure that the probability of exploration (selection of services for which there is little or no prior data in the model) is linked to the relative improvement expected when exploration is pursued over exploitation (selection of those for which there is a large amount of data). An adaptive momentum mechanism for updating the model has been developed so that the incorporation of new data into the model is dependent on the amount and recency of the information already stored. The methods used allow a system of multiple agents to be adaptive to changes in the service environment improving the overall QoS of the system, and may be made more effective through introducing collaborative strategies.

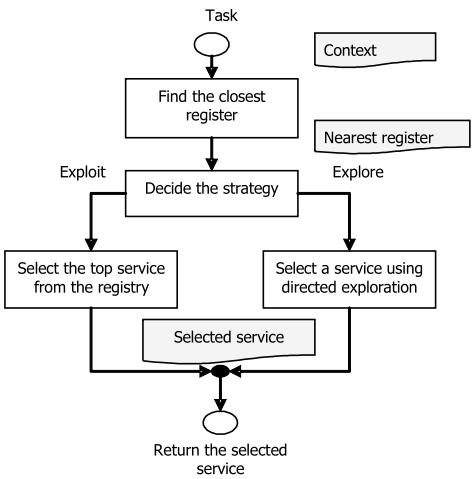

FIGURE 2. Exploration/Exploitation, Decision.

Two collaborative gossiping strategies have been investigated which vary in the degree to which the selector agents share information. The first strategy, anonymous gossiping, involves only partial sharing of information and allows selector agents to gain a better estimation of the distribution of QoS attainable in the network on which the exploration-exploitation control is based. The second collaborative strategy, *full gossiping*, involves sharing detailed information about providers between selectors to speed up learning through exploration. The agents, although cooperative may, however, choose to be selective with the information about providers which they share with others so as not to create unfavourable competition on a subset of service providers, and hence undermine their own performance - *secretive full gossiping*.

The task processing cycle is illustrated in Fig. 3 whereby a task is dispatched to the selector agent and based on both the results of functional service discovery and the selection model built up so far, a service is selected to process the task. The QoS with relation to the task is calculated and used to either *augment* the model if the chosen service was not experienced in the past, or *adapt* the model in the case that there was past experience.

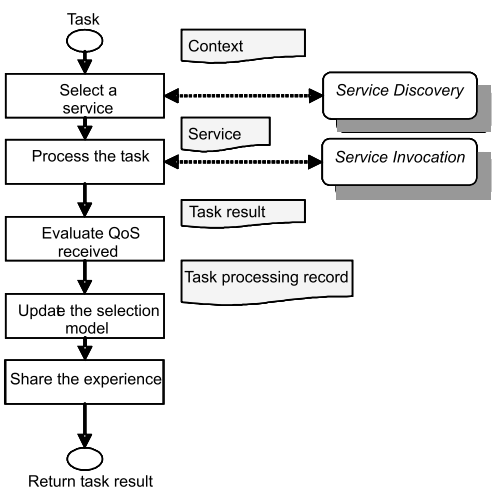

FIGURE 3. Task process cycle.

The selection model consists of registers which represent clusters of experience for services used for particular tasks and contexts and are used to simplify the problem space. An important autonomous decision that the selector agents must make is whether to exploit their existing (usually incomplete) model and choose the service which they expect will act best or explore the service landscape further and either select a service for which there is a sub-optimal expectation of QoS or for which there is no prior experience. In Mercury, the calculation of the expected gain from exploration is distributed by making the agents collaboratively share

their expected outcomes. This ensures that agents have a reliable understanding of the distribution of QoS achievable throughout the network of services, improving their decision-making ability of whether to explore or not. Further details of the Mercury framework and similar approaches in literature are discussed in depth in [6]. This includes the defined structure of the model and the precise algorithms involved with its construction and adaptation.

3.2. Related Work

The majority of related work addresses the problem of service selection based on QoS by formalising the QoS requirement space. This is often achieved by defining a QoS ontology [11, 20-22] which is used to specify the qualities that constitute QoS. This is then used for a consumer of a service to specify strict requirements as well as advertising certain quality capabilities from the provider side. Mercury addresses the need for more work dealing with ranking services based on their QoS without explicit QoS requirements as suggested in [19].

The notion of providers advertising their own quality capability, however, brings about the question of trust, which is dealt with by the related works through trust and reputation modelling, in particular [11, 17]. Trust can be used to build relationships between consumers and providers explicitly based on reputation; however the alternative approach taken by Mercury is to build up relationships implicitly based on agent learning dynamics and their interactions with other agents, the details of which make up the main contribution of this work.

A small number of works acknowledge that defining quality requirements and capabilities using precise terms is not always suitable and that instead fuzzy terms of quality may be adopted [4, 20]. It is envisaged that the Mercury framework will be extended to adopt this approach; however, this is left for future work.

In [22], a multidimensional QoS model similar to that of Mercury is presented, although Mercury goes further to propose a mechanism for which this model can be populated in a collaborative fashion using a multi-agent system.

Much of the work surveyed adopts a decentralised (P2P) architecture combining agents to perform collective modelling. Of particular relevance are [3] and [11]: in the former, a reputation model is built based on peer votes for quality; whilst in the latter, quality ratings are shared via rendezvous nodes in the network. Sonnek et al. [15] have developed and evaluated a task allocation mechanism based on statistical modelling of provider reliability. In contrast to our approach, they use a central reputation server, and they do not consider competition between the clients of the allocation mechanism.

We have previously conducted work using a more theoretical approach whereby relationships with service providers are established based on past experience and simple rules which cause emergent self-organisation of peers [14]. Mercury builds on this work by adding task and context-aware capabilities in the internal model of service selection and by introducing the notion of designated selector agents which may collaborate in order to further improve their selection behaviours.

3.3. Experimental Analysis

In order to quantitatively compare the main features of the Mercury framework a simulation environment has been developed which can be populated with n providers of a single service and m service selector agents. We abstract away from the notion of consumers in this case and assume that both the task and context parameters of the problem stay constant.

We were particularly interested in investigating the effectiveness of the system in the case where QoS of a particular service degrades depending on how many simultaneous connections there are to it at any one time. In this sense there is competition for resources and in order to reach an optimal configuration of service selection, it is necessary for the selector agents to both be able to form relationships with certain providers whilst remaining adaptive to changes. In the simulation, the environment is dynamic in the sense that resultant QoS is non-deterministic from an individual selector's point of view due to competition and the distribution of QoS capability can be parameterised.

For all of our experiments the simulation was set up with 30 service providers and 5 selector agents and consumers. The QoS capability distribution was set to uniformly increase such that the 1^{st} provider had the minimum capability and the 30^{th} provider had the maximum (zero and one, respectively). At each time step in the simulation, each selector agent chooses a provider to be invoked and receives the measure of QoS from the provider as a result. The internal selection model is built up through subsequent time steps and at the end of each time step, each selector agent may gossip with other selector agents, depending on their gossiping strategy. The results are averaged over 10 runs.

The first set of experiments was used to compare the different selector agent collaboration strategies on the resultant system (global) QoS attained (Figure 4). It is clear that gossiping enables the QoS to be increased faster and rather unsurprisingly full gossiping produces the fastest rate of QoS increase through the initial stages. The full gossiping approach would be highly effective if at some point the service landscape were to change dramatically. With little or no provider churn, though, full gossiping actually results in a lower QoS than if there was no communication. This demonstrates how, by sharing information about the "best" provider with other agents results in unfavourable competition whereby relationships between a selector S_1 and a particular provider P becomes infected by another selector S_2 which has gained information about P from S_1 and so believes that such a relationship is best for it too. In this case, the global QoS actually decreases. Secretive full gossiping aims to counteract this effect by not sharing "best" providers between selector agents. Indeed, Fig. 4 indicates that the resultant QoS is highest when using the secretive full gossiping strategy. A slight lag compared to the full gossiping curve can be seen and this represents the trade-off of not sharing with other agents the top provider. The secretive full gossiping strategy also clearly performs best in the aggregate performance comparison (Table 1), which takes into account both the resulting level of QoS and the speed with which it is

achieved. The anonymous gossiping strategy clearly also proved to be very good but elicits slower convergence which demonstrates that there is a case for sharing direct references to providers such as in the full and secretive strategies. Nevertheless, its effectiveness highlights the importance of collaborating to improve the data on which the exploration/exploitation decision is based.

Collaborative strategy	*Aggregate performance*
No communication	0.65
Anonymous gossiping	0.69
Full gossiping	0.65
Secretive full gossiping	0.71

TABLE 1. Average aggregate effect of different selector agent collaboration strategies on resultant system QoS derived by averaging each of the 25-cycle sequences.

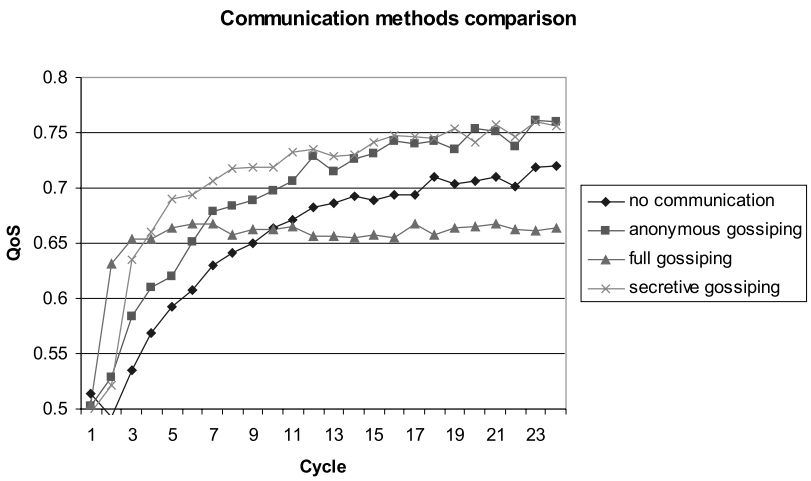

FIGURE 4. Effect of different selector agent collaboration strategies on resultant QoS.

The second set of experiments set out how the adaptive exploration probability mechanism employed by Mercury compared to a fixed strategy. For all the experiments the secretive full gossiping strategy was used although the other strategies produced similar results when tested.

Fig. 5 shows the results from this second experiment set and shows that the adaptive exploration mechanism is particularly useful in the initial stages where

little is known about the services available. It also results in a level of QoS almost as good, as the best fixed level of exploration found a value of $\varepsilon = 0.2$ in an ε-greedy strategy [16] resulting in the highest average QoS. The main use of the adaptive exploration, though, is the adaptivity which it gives the system, allowing the selector agents to choose the appropriate amount of exploration given the conditions in the network and the accuracy of their selection models, rather than performing "blindly" following a fixed probability of exploration, or perhaps a pre-defined exploration-exploitation scheduling function.

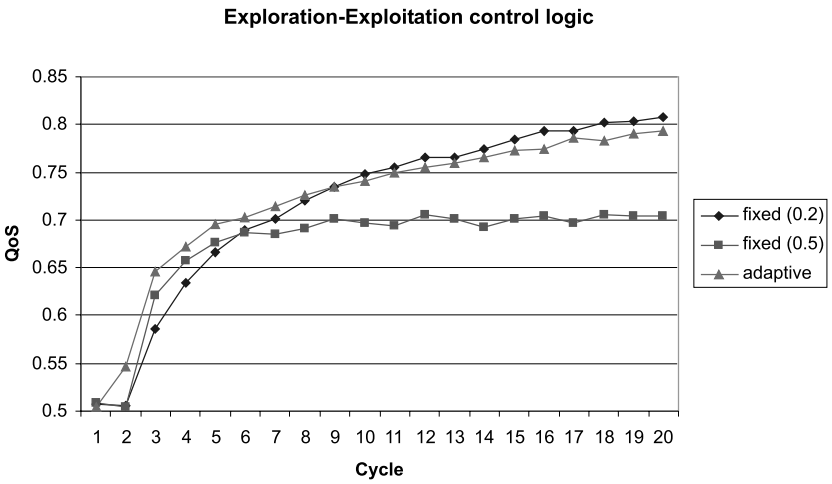

FIGURE 5. Mercury adaptive vs fixed probabilistic (ε-greedy) strategies.

Fig. 6 is an illustration of how the probability of exploration changes over time on average in the experiments. In casing like the experimental set-up, where the QoS data remains relatively static over time, we see an exponential decrease of the likelihood of exploration as the selector agents become increasingly confident with their model that they're building up and the relative advantage of exploring new services over exploiting those deemed best decreases. In a more dynamic scenario we'd see this graph peak at times of change of the service landscape where better services are introduced or perhaps some existing services are able to offer a higher level of QoS. The peaks would signify the selector agents reacting to this change in the service landscape accordingly and the potential for increased exploration would quickly spread throughout the population by means of (anonymous) gossiping.

The Mercury framework is a concrete illustration of how emergent properties can be leveraged to improve global system behaviour in Service-Oriented Architectures, such as Nexus. The combination of local decision-making (exploration/exploitation strategy) with diffusion of QoS information (gossiping) allows a population of selectors with variable needs to collectively identify and converge toward a configuration that meets the requirements of a majority of participants.

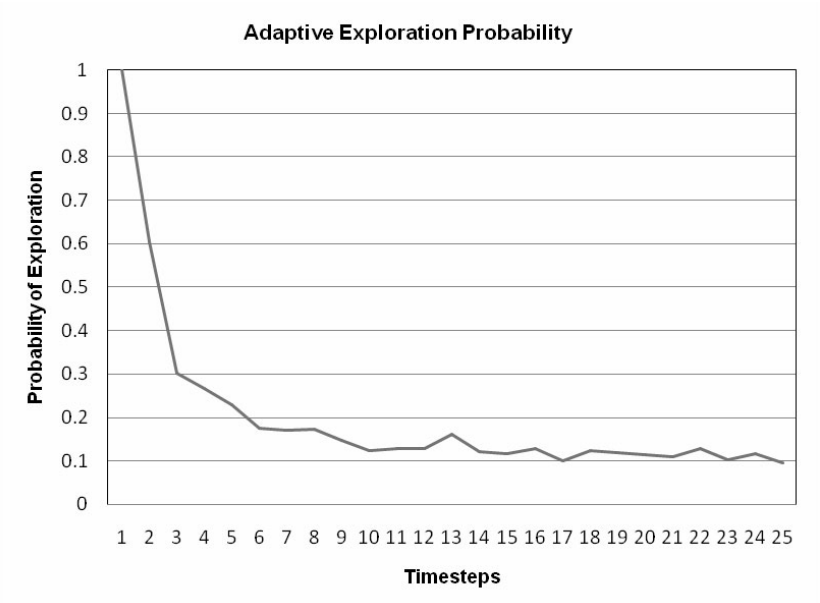

FIGURE 6. Exponential decrease of exploration probability over time as a result of the adaptive mechanism.

Moreover, this distributed problem-solving is largely implicit: the establishment of preferential relationships between selectors and providers incorporates any bias associated with initial conditions and/or the influence of the early history of the system. For instance, in the case that there is competition between two or more selectors for a contended resource, the progressive gain of momentum will ensure that random fluctuations are amplified to the point where only an adequate subset of all competing selectors keep their affiliation with the service. By forcing the 'losers' to identify an alternative provider, this process usually leads to improved global QoS, without any need for central planning or explicit negotiations between selectors.

4. Conclusion

Within the defence domain the problem of data overload continues and will be greatly magnified by the arrival of new high bandwidth sensor arrays and persistent surveillance systems. In addition the lack of skilled IT support manpower makes the problem particularly acute in the defence sector. The Nexus platform is an attempt to merge the best of autonomic computing and agent-based techniques to create a self-organising and self-healing service delivery capability. The result combines the resilience of P2P networks with the service management and legacy

integration power of SOA approaches. The resulting architecture is intrinsically scalable, robust and can be applied at the tactical, operational and back-end layers of deployment. Current development is now focused on integrating new capabilities for ontology management, 3D scenario visualisation, and embedded security for the network itself. These activities are part of the wider cluster of projects within the DIF DTC [2] termed Hyperion.

The race to achieve Network Enabled Capability (or NCW) is a grand challenge endeavour which can only be realised through the application of autonomous agents and self-* system approaches, such as Mercury adaptive service selection. The Nexus platform demonstrates some of the promises such systems can provide.

References

[1] A. Alston, "Network Enabled Capability - The Concept," *Journal of Defence Science* 8(3), 108-116, 2003.

[2] Data and Information Fusion Defence Technology Centre (DIF DTC), www.difdtc.com

[3] F. Emekci, O. Sahin, D. Agrawal and A. Abbadi, "A Peer-to-Peer Framework for Web Service Discovery with Ranking," in *Proceedings of the IEEE International Conference on Web Service (ICWS'04)*, Washington DC, USA, 2004, p. 192.

[4] C.- L. Huang, C.- C Lo, Y. Li, K.- M Chao, J.- Y Chung and Y. Huang, "Service Discovery through Multi-Agent Consensus," in *Proc. of IEEE Int. Workshop on Service-Oriented System Engineering (SOSE'05)*, pp. 37-44, 2005.

[5] M. N. Huhns, M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing* 9(1), pp. 75-81, 2005.

[6] M. Jakob, A. Healing, F. Saffre, "Mercury: Multi-Agent Adaptive Service Selection Based on Non-Functional Attributes," to appear in *Proc. of the 2nd International Workshop on Engineering Emergence in Decentralised Autonomic Systems*,

[7] M. Jakob, N. Kaveh and R. A. Ghanea-Hercock, "Nexus - Middleware for Decentralized Service-Oriented Information Fusion," in *Proc. of Specialists' Meeting on Information Fusion for Command Support*, The Hague, Nov 2005.

[8] Java Message Service, http://java.sun.com/products/jms/

[9] N. Kaveh, R. Ghanea-Hercock, "NEXUS: Resilient Intelligent Middleware," *BT Technology Journal*, 22(3), pp. 209-215, 2004.

[10] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, 36(1), pp. 41-50, 2003.

[11] E. M. Maximillen and M. P. Singh, "Multiagent System for Dynamic Web Services Selection," in *Proc. of the AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE)*, Utrecht, July 2005.

[12] R. Mondejar et al., "Towards a Decentralized p2pWeb Service Oriented Architecture," in *Proc. of 2nd Int. Workshop on Collaborative P2P Information Systems (COPS 2006)*, Manchester, UK, 2006.

[13] M. Paolucci and K. Sycara, "Autonomous Semantic Web Services," *IEEE Internet Computing*, 7(5):34-41, 2003.

[14] F. Saffre and H. R. Blok, "SelfService: A theoretical protocol for autonomic distribution of services in P2P communities," in *Proc. of 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Maryland, April 2005, pp. 528-534.

[15] J. Sonnek, M. Nathan, A. Chandra and J. Weissman, "Reputation-Based Scheduling on Unreliable Distributed Infrastructures," in *Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[17] W.T. Teacy, J. Patel, N.R. Jennings and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, 12(2), 2006.

[18] P. Van Roy, A. Ghodsi, S. Haridi, J.- B. Stefani, T. Coupaye, A. Reinefeld, E. Winter, R. Yap, "Self Management of Large-Scale Distributed Systems by Combining Peer-to-Peer Networks and Components," *CoreGRID Technical Report*, TR-0018, 2005.

[19] L.- H. Vu, M. Hauswirth and K. Aberer, "Towards P2P-based Semantic Web Service Discovery with QoS Support," in *Proc. of Workshop on Business Processes and Services (BPS)*, Nancy, France, 2005.

[20] P. Wang, K.- M Chao, C.- C Lo, C.- L Huang and Y. Li, "A Fuzzy Model for Selection of QoS-Aware Web Services," in *Proc. of IEEE International Conference on e-Business Engineering (ICEBE'06)*, 2006.

[21] X. Wang, T. Vitvar, M. Kerrigan, I. Toma, "Synthetical Evaluation of Multiple Qualities for Service Selection," in *Proc. of the 4th International Conference on Service Oriented Computing*, Springer-Verlag LNCS series, Chicago, USA, December, 2006.

[22] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, 30(5), May 2004.

Alex Healing, Robert Ghanea-Hercock and Hakan Duman
Pervasive ICT Research Centre
British Telecom, United Kingdom
e-mail: alex.healing@bt.com
robert.ghanea-hercock@bt.com
hakan.duman@bt.com

Michal Jakob
Gerstner Laboratory
Czech Technical University
Czech Republic
e-mail: jakob@labe.felk.cvut.cz