In International Journal of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents, World Scientific, Vol. 11, No. 5, pp. 621-641, 2001.

HIGH PERFORMANCE SERVICE DISCOVERY IN LARGE-SCALE MULTI-AGENT AND MOBILE-AGENT SYSTEMS

JUNWEI CAO

DARREN J. KERBYSON

GRAHAM R. NUDD

Department of Computer Science

University of Warwick

Coventry, CV4 7AL, UK

junwei@dcs.warwick.ac.uk

Tel. 442476522863

Fax. 442476573024

Abstract

With increasing requirements of distributed software systems, software agents are becoming a mainstream technology for software engineering and data management. Scalability and adaptability are two key challenges that must be addressed. In this work a new model is introduced for building large-scale distributed software systems with high dynamics, using a hierarchy of homogeneous agents that has the capability of service discovery. The performance of the agent system can be improved using different combinations of optimisation strategies. A modelling and simulation environment has been developed to aid the performance evaluation process. Two case studies are given and simulation results are included that show the impact of the agent mobility and the choice of performance optimisation strategies on the overall system performance.

Keywords: Service discovery; performance evaluation; multi-agent systems; mobile agents; modelling and simulation.

1. Introduction

Software agents are becoming a more and more important software development technology. The key sign of this trend is the emergence of diverse applications and approaches in many different areas [3]. To build large-scale multi-agent and mobile-agent systems, scalability and adaptability are two key challenges that must be addressed.

- ***Scalability***. The performance of a multi-agent system may decrease as the number of the agents in the system increases. The system cannot scale well when some agents may potentially become system bottlenecks on computing or communication. For example, if agents in a system are not aware of each other and predefined to work together, it is important for these agents to find and cooperate with each other efficiently even when the number of the agents is very large.
- ***Adaptability***. The agent can change, from time to time, its identity, functions, interface etc. When an agent moves from one host to another, previous identity will not work any longer. The functions and performance of an agent can also vary when its resource and environment change. High dynamics of the multi-agent system makes the agent coordination and cooperation much more difficult.

Many tools and infrastructures have been implemented to aid the development of multi-agent and mobile-agent systems. Their models and motivations vary from each other. The most recent works include HiMAT [14], Jackal [13], OAA [25], SIM_AGENT [29], and JAFMAS [10]. A survey on Java-based agent development environments can be found in [2]. However, none of them focus on the problem of system scalability and agent coordination.

Some research on coordination models and languages [23,11] have been introduced recently to overcome agent coordination problems. In many works described in [12], agents are organised into a hierarchy to address the problem of system scalability, which is also used in our work. These works focus on the data processing or event control [26], rather than knowledge exchanging, among agents. Performance issues have not been a key consideration in their implementation.

There are many other distributed computing and communication infrastructures, e.g. Bluetooth [4], HAVi [22], Jini [1], Salutation [27], UPnP [30] etc. They provide protocols for service advertisement and discovery to coordinate the behaviours of different components in the system, such as SDP (Service Discovery Protocol) [24] in Bluetooth, lookup service [21] in Jini, SSDP (Simple Service Discovery Protocol) [17] in UPnP and SLP (Service Location Protocol) [18] in Salutation. A good survey on service discovery techniques can be found in [28].

Software agents have been accepted to be a powerful high-level abstraction for the modelling of distributed software systems [20]. An agent can be also considered as both a service provider and a service requestor. Agents can work together by federating other agents that can provide the required services [19]. The function implementation of the multi-agent system can be abstracted to the processes of service advertisement and service discovery at a high level. However, performance issues arise when the dynamics of agents and services in the system increase.

Performance optimisation has been discussed in some of the implementation of large-scale distributed systems like DNS (Domain Name Systems). However, most of existing large-scale systems are basically static and system performance can be only evaluated qualitatively. Our research described in this work focus on quantitative performance evaluation of service discovery in large-scale agent systems with highly dynamic behaviours, which grew out of experiences with two agent-based systems starting with MACIP-OAS [5,6], an early attempt to integrate manufacturing software using operational administration agents [15], and continuing with GRID-ARMS [7,8], an agent-based resource management system for grid computing [9].

In this paper, a new model is presented for building large-scale distributed software systems with high dynamics using a hierarchy of homogeneous agents, which has the capability of service discovery. Different optimisation strategies can be used to improve the system performance according to predefined metrics. A performance modelling and simulation environment has been developed that enables the performance of the system to be investigated quantitatively. Two case studies under the mentioned application backgrounds are given and simulation results are included that show the impact of the agent mobility and the choice of performance optimisation strategies on the overall system performance. The rest of the paper is organised as follows:

- In Section 2 a common model of the agent system is presented;
- In Section 3, the metrics used for the performance evaluation of this hierarchy, and the different optimisation strategies available are described;
- In Section 4 the simulation environment is described and results from two case studies are presented; and
- Preliminary conclusions and ongoing works are discussed in Section 5.

2. System Models

In this section, a common agent structure is given firstly, followed by an explanation of how agents can be organised into a hierarchy. This is the basis of understanding the service advertisement and discovery mechanisms described in next section.

2.1. *Agent structure*

There is a single type of the component, the agent, which is used to compose the whole system. Each agent has the same set of functions and acts as a manager of local services. However, at a high level, an agent must also take part in the cooperation with other agents. A layered agent structure is shown in Figure 1 and described in detail below.

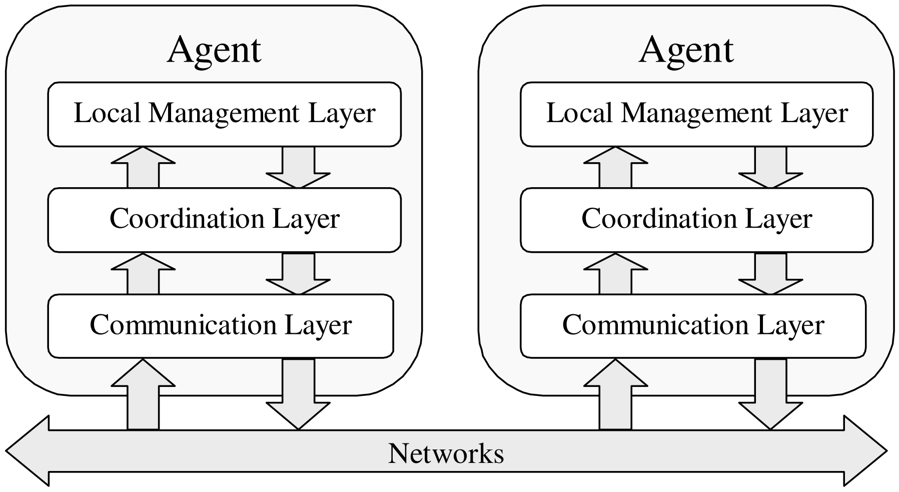


Fig. 1. Agent structure.

- ***Communication Layer***. Agents in the system must be able to communicate with each other using common data models and communication protocols. Agent Communication Language (ACL) can be used to for agents to exchange knowledge with each other. In some simple systems, pre-defined data structures can also be used instead of a language.
- ***Coordination Layer***. The data an agent receives at the communication layer should be explained and submitted to the coordination layer, which decides how the agent should act on the data according to its own knowledge. For example, in an agent system with service discovery capabilities, the coordination layer of each agent must maintain some kinds of Agent Capability Table (ACT), which is used to record the information on services provided by other agents. The basic structure of an ACT item consists of two parts: agent identity and service information. The agent can look up these knowledge to find a suitable service for a request. As mentioned above, our work focuses on performance evaluation of these processes, which are discussed in greater detail in Section 3.

- ***Local Management Layer***. This layer encapsulates the functions needed for management of local services. For example, if an agent finds that the required service is within its own capability, the relevant information will be submitted to this layer from the coordination layer. This layer should also provide local service information needed by the coordination layer to make decisions.

2.2. ***Agent hierarchy***

The hierarchical model of the agent system is illustrated in Figure 2. Agents can send requests and provide services. Every agent can act as a router between a request and a service. In Figure 2 different terms are used to differentiate the level of the agent in the hierarchy. The broker is an agent that heads the whole hierarchy, maintaining all service information of the system. A coordinator is an agent that heads a sub-hierarchy. A leaf-node is actually termed an agent in this description.

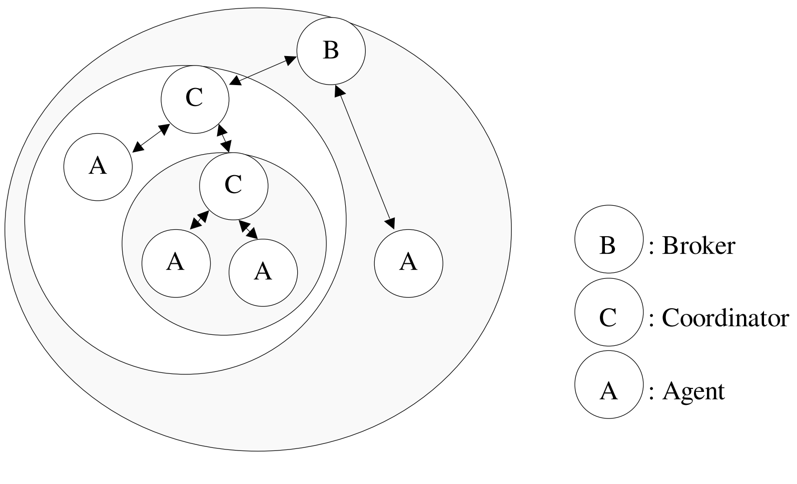


Fig. 2. Agent hierarchy.

When a new agent wants to join the system, in the hierarchical model, it will broadcast to find it’s nearest existing agent. An agent can only have one connection to an agent higher in the hierarchy to register with, but be registered with many lower level agents. All requests that enter a sub-hierarchy must arrive at the coordinator of the sub-hierarchy first and then dispatched to the lower agents. From the view of service providers, a sub-hierarchy can be regarded just as an agent.

If an agent has the required service information, it can contact the target agent directly. Otherwise, it must search its local agents, or ask its upper agent, for a service discovery (i.e. to find an agent that can provide the requested service). The lower or upper agent can also ask other agents for assistance until the service information is found. The agent can then connect directly to the target and directly request the service. All connections between the agents are broken after use of the service is finished.

The services offered by an agent can change over time. When this occurs, the corresponding service information needs also to be updated. An agent can also move from one host to another. When this occurs, the agent identity of corresponding services needs to be updated. The dynamics of the system increases the difficulty of the agent coordination and system management. The essential issue is how an agent advertises its services and also coordinates with other agents to discover the required services in the most efficient way. In the next section, high performance service discovery will be introduced in greater detail.

3. High Performance Service Discovery

In the agent system with service discovery capabilities, there is another important process, service advertisement, which is related to the system performance.

- ***Service Advertisement***. The service information of an agent can be advertised in the hierarchy (both up and down). Different strategies can be used to decide when, and how, to advertise a service but with different performances. This is discussed in greater detail in Section 3.2.
- ***Service Discovery***. When an agent requests a service, it will first check its own knowledge

to see if it is already aware of the service. If it is, it will contact the target agent directly. Otherwise it may contact its upper or lower agents until the available service is found.

There are two extreme situations described in detail below.

- *No service advertisement – results in complex service discovery*. In this situation no ACTs are maintained in the agents. Each agent has no knowledge of the services offered by other agents. When a service is request, a service discovery process is required which may be complex and traverse a large number of agents in the system. The service information is pulled from the agents at discovery time, and so this is a pure *data-pull* model.
- *Full service advertisement – requires no service discovery*. In this situation, each agent advertises to all the other agents. Hence each agent has complete knowledge on the available services in the system and no discovery process is required. When a request is made, the service is found in any agent's ACT. The service information has been pushed to the other agents during the advertising processes, and so this is a pure *data-push* model.

Different systems can use different optimisation strategies to achieve high performance. For example in static systems, where the frequency of change in agent identities and service information is far less than the frequency of service request, the pure data-push model can be used to achieve high performance service discovery. In extremely dynamic systems, where the frequency of agent movements and change in the service information is far greater than the request frequency, the pure data-pull model can be used to achieve high performance. Most practical systems will have characteristics in-between these two extremes. In this section the basic performance issues in the service discovery are introduced. Some common performance metrics are given first and different optimisation strategies are discussed next.

3.1. *Service discovery metrics*

There are kinds of performance criteria that can be used to describe the service discovery performance part of the model. What is considered as high performance depends on the system requirements. However, there are some common characteristics of the system that are usually a concern of the system developer.

3.1.1. *Discovery speed*

Each request from an agent can pass one or more agents in order to find a target agent that can provide the required service. The performance of the discovery process is mainly based on the number of routing connections, since the size of data communication is small. Fewer connections has a quick discovery process, and the higher system performance. In the whole system, there may be simultaneous service requests. The average service discovery speed, *v* is defined as:

$$v = \frac{r}{d} \tag{1}$$

where *r* is the total number of requests during a certain period, and *d* is the total number of connections made for the discovery.

3.1.2. *System efficiency*

The cost for the service discovery also includes connections made for service advertisement and data maintenance. Service advertisement may add additional workload to the system. For each request to find a corresponding service, the total number of connections, *c*, between agents includes those for the discovery processes, *d*, and also those for the advertising processes, *a*.

$$c = d + a \tag{2}$$

The efficiency of the system can be considered as the ratio of the total number of requests, *r*, during a certain period, to the total number of connections *c*.

$$e = \frac{r}{c} \tag{3}$$

3.1.3. *Load balancing*

In some of the systems when the system resources are critical, load balancing may be an important issue. In this system, no agents are used only for service discovery. There is no reason to have any agent with a higher discovery workload than any other. For a system with n agents, the workload, w_k, of each can be described as

$$w_k = o_k + i_k \qquad (k = 1......n) \tag{4}$$

where o_k and i_k are the outgoing and incoming number of connections. The mean square deviation of the w_k can be used to describe the load balancing level of the system, b:

$$b = \sqrt{\frac{\Sigma_k \left(w_k - \overline{w}\right)^2}{n}} \qquad \text{where} \quad \overline{w} = \frac{\Sigma_k w_k}{n} \tag{5}$$

3.1.4. *Success rate*

In some of the performance optimisation strategies the discovery model cannot guarantee to find the target service (that may actually exist in the system). However, in a general system a reasonable service discovery success rate should always be achieved. The success rate, f, describes successful service discovery, using the ratio of the total number of satisfied requests, r_f, during a certain period, to the total number of requests, r:

$$f = \frac{r_f}{r} \times 100\% \tag{6}$$

Most of the time, these service discovery metrics are conflictive, that is not all the metrics can be high at the same time. For example, a quick discovery speed does not mean high efficiency, as sometimes quick discovery may be achieved through the high workload encountered in service advertisement and data maintenance, leading to low system efficiency. It is necessary to find the critical factors of the practical system, and then to use the different performance optimisation strategies to reach high performance.

3.2. *Performance optimisation strategies*

There are several kinds of performance optimisation strategies that can be considered. Most have been used in current practical systems with an assumption that the performance can be optimised. The effects of these strategies are not discussed in detail especially when the dynamics of the system increases. The combination of these strategies may also lead to a different performance.

3.2.1. *Use of cache*

Caching previous service discovery results is a good strategy for performance optimisation that assumes a request may be required more than once. Cached service information is expressed as C_ACT. When an agent sends a request for service discovery, the result can be stored in C_ACT, and hence looked up when next requested. If however the service has changed and is not available any more, the agent may update the C_ACT and perform another service discovery.

Many current network applications use caches to optimise performance. Using cached service information may result in direct service discovery in one step. Another advantage of using cache is that it adds no additional data maintenance workload. However, if the service information changes frequently compared to the request frequency, using cache may decrease the service discovery speed. So the efficiency of using cache depends on the characteristics of the actual system.

3.2.2. *Using local and global knowledge*

Adding some local or global knowledge to an agent is also a performance optimisation that assumes that services are often required by local agents. A request may need fewer connections to find the local service, as the higher-level agents need not take part in the discovery process. The system load can also be reduced.

In order to coordinate the agents to find the services, two kinds of ACTs can be used in each agent to record the service details and information, which are local (L_ACT) and global (G_ACT). Each agent has one L_ACT to record the service information about the agents registered with it. If a request is within the capabilities of the local agents, the agent may directly dispatch the request to the target agent. The G_ACT in an agent is actually a copy of its upper agent's L_ACT. Thus an agent can have the information of more services and be able to contact them directly without submitting the request to the upper agent.

Unlike the C_ACT, additional data maintenance workload is needed for the L_ACT and G_ACT. There are basically three ways to maintain their contents. Firstly, the agent itself can go to pull the corresponding data directly. For L_ACT, the agent can ask its lower agents for their L_ACT, and for G_ACT, the agent can ask its upper agent for its L_ACT. Secondly, the maintenance of the L_ACT and G_ACT can also be driven by changes in service. If contents in one L_ACT of an agent are changed, it may report this to the L_ACT of its upper agent, and may also inform the change to the G_ACTs of its lower agents. Thirdly, the L_ACT and G_ACT can also be updated in the same way as the C_ACT.

The process for the service discovery using the L_ACT and G_ACT is also different from that using the C_ACT. When an agent receives a request, it will look up its L_ACT. If the agent finds that one of its lower agents can provide the service, it will dispatch the request directly to the corresponding agent. Otherwise, it will look up its G_ACT. If G_ACT shows that another agent can provide the service, it will dispatch the request to that agent. If the service is not found in either, the agent will ask its upper agent for further service information. After the upper agent returns the result, it can update its own G_ACT and return the result to the agent who originated the request.

3.2.3. *Limit service lifetime*

Another performance optimisation strategy is adding a service lifetime limitation to the attributes of the service information. This lifetime should be pre-estimated before the service is advertised. The agent can check the ACTs frequently and delete out-of-date service information. This can avoid unnecessary routing processes and increase the speed of service discovery. There is also no additional data maintenance workload. However, the lifetime of some services in the system may be unpredictable.

3.2.4. *Limit scope*

The scope in which a service can be advertised and discovered can also be pre-defined by attributes to the service information. The service need only be advertised within a certain scope of the system, which can reduce the advertisement and data maintenance workload. The search for a service can also be limited to a certain scope avoiding unnecessary discovery processes. However, pre-knowledge about the service and its requests are needed to achieve optimisation. Mismatches between the scope limitation of a service and of a request may result in the low success rate of the service discovery.

4. Performance Evaluation

In the previous sections an agent hierarchy that has a service discovery capability has been described. This can be used to give a model of large-scale distributed software systems with highly dynamic behaviours. However, performance evaluation of such a system is a difficult task, because the system behaviour will become complex when different optimisation strategies are used. In this section, a modelling and simulation environment is introduced with two case studies.

4.1. *A modelling and simulation environment*

There are four kinds of information that effect the system performance and must be defined in the performance model. These include: the agent hierarchy, the services, the requests, and the optimisation strategies. The modelling and simulation environment provides graphical interfaces for the user to perform the modelling activity at both the agent level and the system level. The only components that exist in the model are agents, so agent-level modelling can be used to define all the model attributes for the simulation. However, system-level modelling is also necessary for modelling agent mobility, service and request distribution, and so on. A system-level strategy definition can affect all of the agents in the model and ease the modelling process.

When the simulation begins, a thread is created to calculate the statistical data step by step. The phase for request sending and the service discovery is the key part of the whole simulation process. The simulation environment can show the results in multiple views, including a step-by-step, accumulative view etc. The output data are used to provide full support for all performance metrics. These are also illustrated in detail in Figure 3.

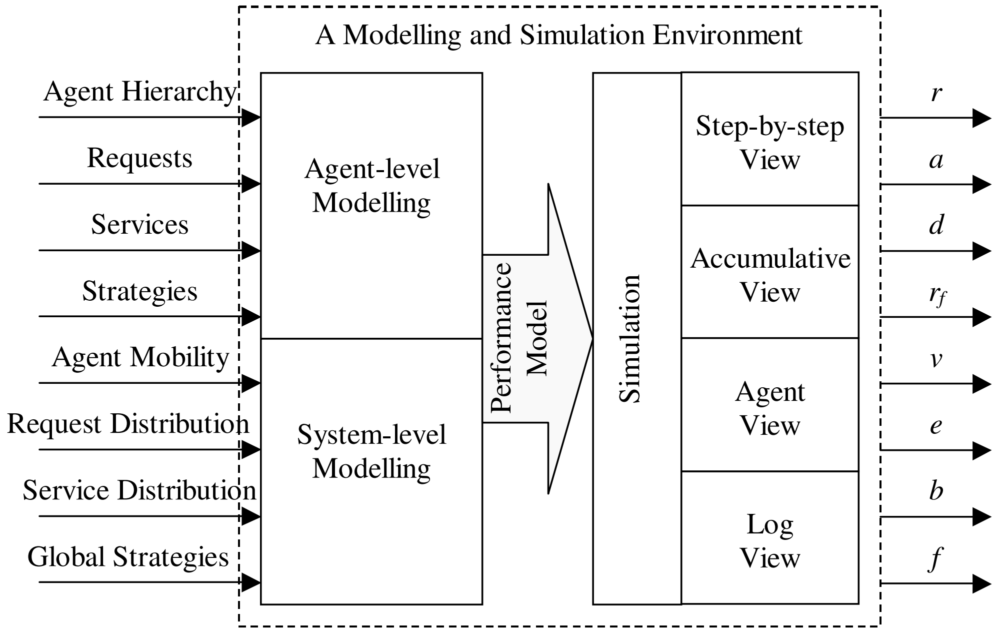


Fig. 3. Performance modelling and simulation.

4.2. ***Case study I – impact of agent mobility on service discovery performance***

Agent mobility has important effect on the system performance. When an agent moves from one host to another, other agents in the system need to spend more time on looking for the services it provides. Using proper optimisation strategies can reduce the system workload on service discovery and improve system performance. The case study introduced in this section is abstracted from our previous works on MACIP-OAS.

4.2.1. *Application background*

CIMS Application Integration Platform (MACIP) is developed to offer manufacturing enterprises with a complete solution for the CIMS implementation through integrating a set of application software products [15]. Operational Administration System (OAS) is the kernel of the MACIP to implement integration functions [5]. Dynamic application integration is essential for the OAS to support openness, scalability and maintenance of the MACIP system.

Multi-agent technology is used in OAS to implement the integration of different software applications. Each agent is wrapped with one or more applications and takes these applications as services that can be provided to other agents. The communication and cooperation among these applications are implemented via service discovery among the agents. Applications may be added to or remove from the system at any running time. Agents must be flexible enough to adapt to these dynamic behaviours of the system. The example model described below is abstracted from the MACIP-OAS. Some experiments have been done using the modelling and simulation environment and results are given to show the impact of agent mobility on service discovery performance of agent systems with different optimisation strategies.

4.2.2. *Performance model*

A simple multi-agent system model is shown in Figure 4, containing 26 agents. The whole system is configured to have only one service named *Print*. The agent that can provide the service is *Printer* now connected to *till* and later, during the simulation, is moved to connect to *sun* with a new identity *NewPrinter* (this is not shown in Figure 4). All the other agents may or may not request the *Print* service with different frequency (Note that the details of requests are not given below).

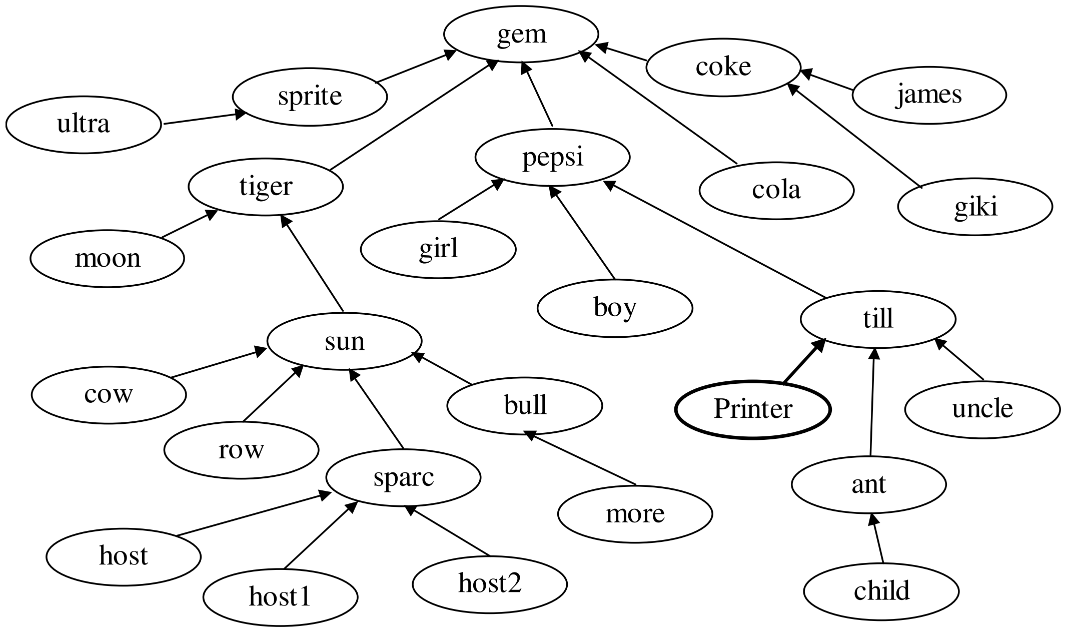


Fig. 4. Example model (I): agent hierarchy.

Two experiments are used to show the impact of agent mobility on the service discovery performance with different optimisation strategies. The strategies are only defined at the system level, which means all of the agents in the model must use the same performance optimisation strategies. In experiment No. 1, only L_ACT is used, while the L_ACT and G_ACT are both used in each agent in experiment No. 2. L_ACTs are maintained by the real-time service advertisement. G_ACTs are updated once every 30 steps. The mentioned agent movement will happen at the 100th simulation step.

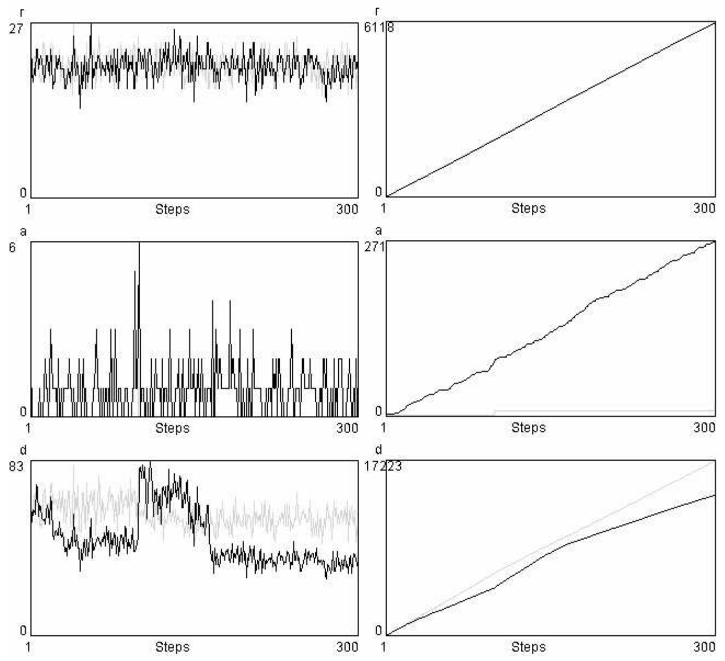
Table 1. Example model (I): choice of strategies.

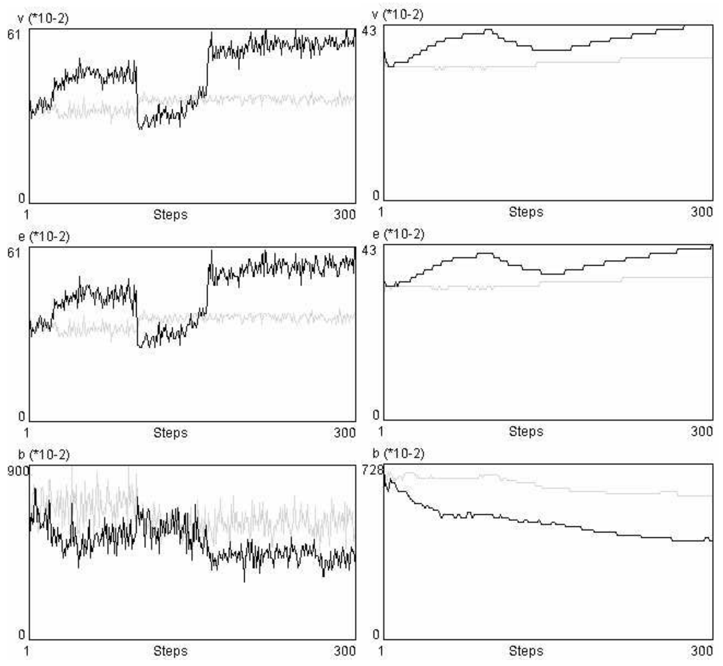
Optimisation Strategy	Experiment Number	
	1	2

Using C_ACT	-	-
Using L_ACT	√	√
Using G_ACT	-	√
Updating L_ACT	-	-
Updating G_ACT	-	√*
Advertising L_ACT	√	√
Multicasting L_ACT	-	-

* Here the updating frequency was once every 30 steps.

4.2.3. *Simulation results*





Note: —— for Experiment No. 1; —— for No. 2.

(a) Step-by-step view (b) Accumulative view

Fig. 5. Simulation results (I).

Figure 5 shows the simulation results for 300 steps. A step can be designed as an arbitrary number of seconds. The results of experiment No. 1 show that the agent mobility has little impact on the service discovery performance in this situation. By using G_ACTs in experiment No. 2, the system performance is improved, which includes quicker service discovery, higher system efficiency, and better workload balance. Compared with experiment No. 1, the simulation process of experiment No. 2 is more complex when the agent movement happens. The discovery success rate is assumed to be not critical and attention is not given to it in this study. The whole process can be divided into five phases, which are explained in detail below.

- *Learning phase*. In the first 40 steps, the G_ACTs of the agents are updated gradually and more service advertisement leads to increasing discovery speed and system efficiency, and better workload balance. This can be viewed as an agent learning process.

- *Stable phase*. After about 40 steps, the curves begin to be flat. All G_ACTs of the agents have been updated and there are no service changes, so the system runs in a stable mode with high performance.
- *Agent mobility*. The defined agent mobility happens at the 100th simulation step. When the agent moves it must advertise to remove its service information from the old agent hierarchy and to add the new service information to the new agent hierarchy. This causes the increase of the connections for service advertisements (a). The service information in all the agents becomes out-of-date, which results in more workload for the service discovery (d). So the average service discovery speed (v) and system efficiency (e) decrease greatly. During a couple of steps after the agent movement, the system performance in experiment No. 2 is even worse than that in experiment No.1. This is because the out-of-date information in G_ACTs may misguide the service discovery processes.
- *New learning phase*. This phase is the same as the previous learning phase. The agents learn about the new identity of the service *Print* gradually via the G_ACT updating.
- *New stable phase*. The agent mobility leads to a stable mode with higher performance finally. This is because *sun* is the coordinator of a larger sub-hierarchy than *till* is. When the service is moved, more requests become local instead of remote, which reduces the discovery workload of the system. In fact, in experiment No. 1, the system enters the new stable phase immediately after the agent moves. The slight performance improvement also indicates the impact of the hierarchy itself on the system performance.

In order to give a clear explanation of why system workload balance can be improved by using G_ACTs, the workload view of agent *gem* is illustrated in Figure 6. Agent *gem* is the head of agent hierarchy in this example model. In experiment No. 1, the workload for service discovery of *gem* is rather high, because each time a lower agent has no information of the required service, it submits the request to *gem*. The situation is changed when G_ACTs are added to agents. Each lower agent maintains a copy of L_ACT of *gem*, which can be used directly without submissions. This keeps *gem* a low workload for service discovery in stable mode of the system and *gem* only takes part in service discovery processes in some dynamic situations, say, when agent movement happens. The workload reduction of the broker and coordinators in the agent system leads to the better load balance of the system.

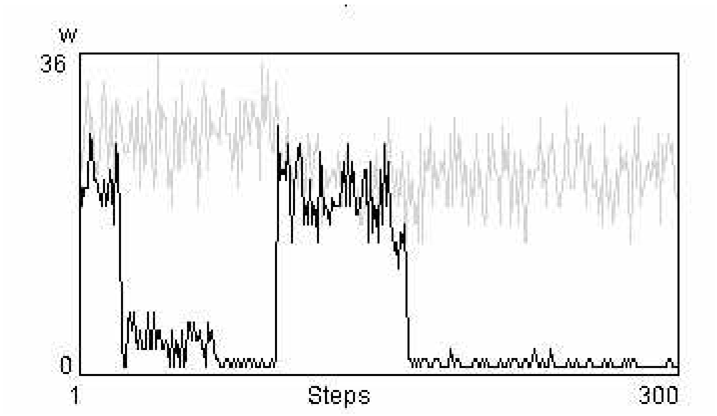


Fig. 6. Service discovery workload of *gem*.

This is a small example model with only one agent movement. The system model is not a large-scale one and the service in the system is static during most of the simulation time. In fact, if there are several agent movements in the example model, introducing G_ACTs to the system may result in less performance improvement. A large-scale multi-agent system model with highly dynamic services is used in the next case study.

4.3. *Case Study II – impact of choice of strategies on service discovery performance*

The impact of choice of strategies on service discovery performance depends on the real situation of the system dynamics. In this case study, the dynamic behaviours of the system lie in a frequent change of service performance, instead of agent mobility.

4.3.1. *Application background*

Computational Grids are software infrastructures to provide dependable, consistent, pervasive, and inexpensive access to high-end computational capability [16]. The overall aim of our resource management system, GRID-ARMS, is to efficiently schedule applications that need to utilise the available resources in the grid environment [9].

An agent-based hierarchical model is used in GRID-ARMS and resource management, scheduling, and allocation can be abstracted to the processes of service advertisement and service discovery. However, a grid is a dynamic environment where the performance of the resources are constantly changing. Agent system should be able to choose proper strategies to support service discovery efficiently.

4.3.2. *Performance model*

The attributes of an example model are shown in Table 2. This is composed of about 250 agents, each representing a high performance computing resource that may provide a computing capability with a different performance. These agents are organised in a hierarchy, which has three layers. The identity of the root agent is *gem*. There are 50 agents registered to *gem*, four of which each also have 50 lower agents. The hierarchy is illustrated in Table 2(a).

Table 2. Example model (II)

Agents	Upper Agent
gem	-
sprite~0……sprite~49	*gem*
tup~0……tup~49	*sprite~9*
cola~0……cola~49	*sprite~19*
tango~0……tango~49	*sprite~29*

pepsi~0……pepsi~49	*sprite~39*

(a) Agent hierarchy

Name	Relative Performance	Freq.	Lifetime	Scope	Dist (%)
HPC	1000	5	Unlimited	Top	10
HPC	600	10	Unlimited	Top	20
HPC	200	20	Unlimited	Top	30

(b) Services

Name	Relative Performance	Freq.	Scope	Dist. (%)
HPC	100	5	Top	80
HPC	300	10	Top	60
HPC	500	20	Top	40
HPC	800	40	Top	20
HPC	1000	60	Top	10

(c) Requests

Optimisation Strategy	Experiment Number		
	1	2	3
Using C_ACT	√	√	√
Using L_ACT	-	√	√
Using G_ACT	-	√	√
Updating L_ACT*	-	-	√
Updating G_ACT*	-	√	√

Advertising L_ACT	-	√	√
Multicasting L_ACT	-	-	√

* Here the updating frequency was once every 10 steps.

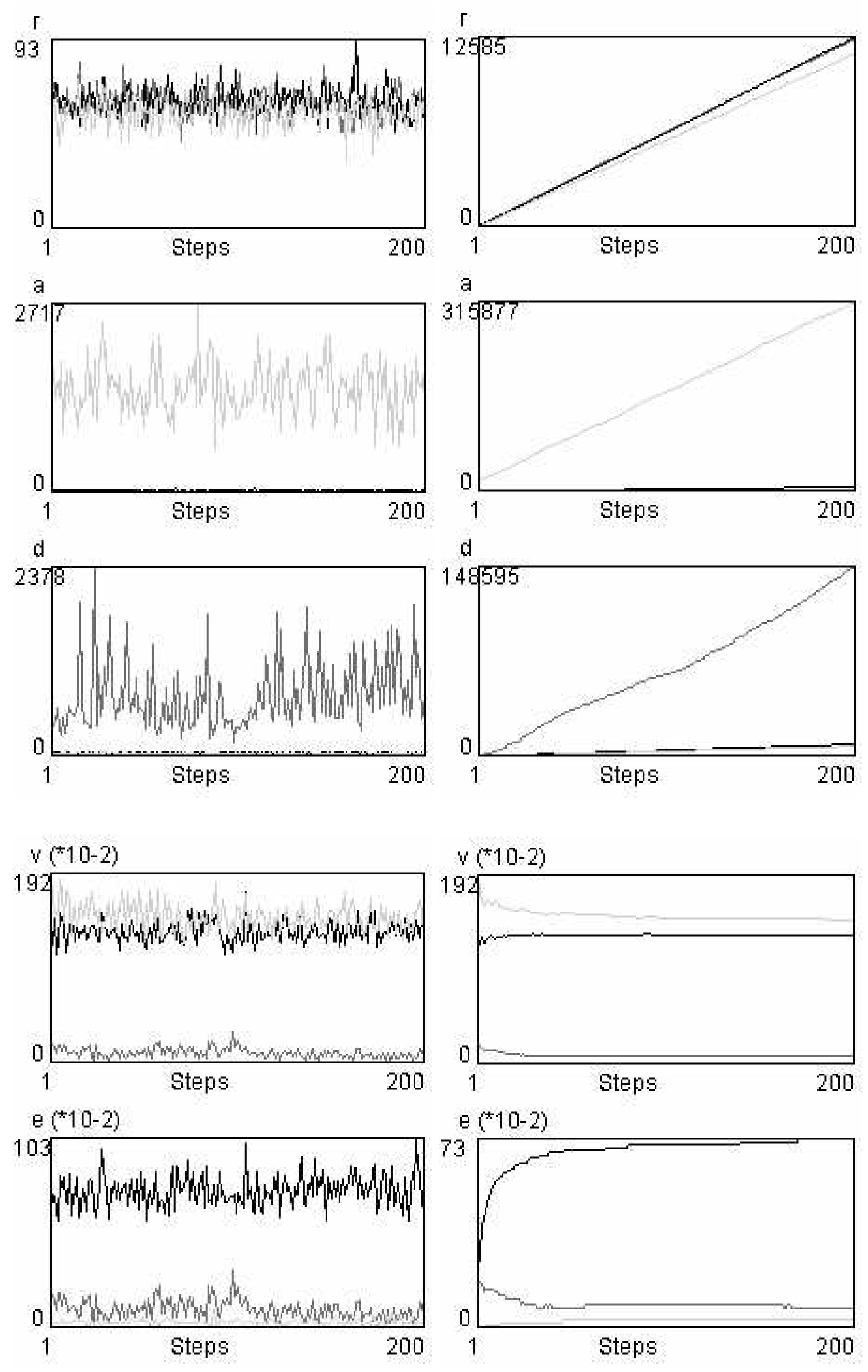
(d) Choice of strategies

To simplify the modelling processes, the services and requests are defined in the agents at the system level. The name of the services and requests are all *HPC*, but with different relative performance values. The frequency value of the service, 5, for example, means the service performance will change between 0 and the performance value once every 5 steps during the simulation. The frequency value of the request, 5, for example, means a request will be sent once every 5 steps during the simulation. The performance optimisation strategies of the lifetime and scope limitations are not used in the model. The distribution value is used to define how many agents will be configured with the corresponding service or request. The simulator will choose agents randomly to be configured with these system level definitions before simulation begins.

Finally, the model must define how each agent uses the cache, local, and global knowledge to optimise the performance. In this case study three experiments have been considered, each of which has the same configuration, but has different optimisation strategies as described in Table 2(d). To simplify the experiments, the strategies are only defined at the system level. A mixture of optimisation strategies is possible but is not considered in these experiments.

4.3.3. *Simulation results*

In the simulation results included in Figure 7, a comparison of the different strategies is given by considering their impact on the system performance. The load balancing and discovery success rate are assumed to be not critical in this study. Attention is given to the discovery speed and the system efficiency. Each of the three situations is described in detail below.



Note: —— for Experiment No. 1; —— for No. 2; —— for No. 3.

(a) Step-by-step view (b) Accumulative view

Fig. 7. Simulation results (II).

In the first experiment, only the cache is used in each agent, which needs no extra data maintenance and the discovery speed and system efficiency are both rather low. This is because the dynamics of the services reduce the effects of the cached information. Each time the request arrives, a lot of connections must be made and traversed in order to find the satisfied service when the cached service information become unreliable.

L_ACT and G_ACT are added in each agent in the second experiment. Each time the service performance changes, the corresponding agent will advertise the change upward in the hierarchy. Each agent will get a copy of the L_ACT of its upper agent once every 10 simulation steps. These add additional data maintenance workload to the system, which decreases the discovery workload extremely. So the discovery speed and the system efficiency are all improved greatly.

More maintenance of the L_ACT and G_ACT are added in the third experiment. Each agent copies the L_ACTs of its lower agents once every 10 steps. The change of the L_ACTs will also be passed to the G_ACTs of the lower agents. These improve the discovery speed only a little, but adds further data maintenance workload, which decreases the system efficiency extremely.

In summary, the strategies used in the second experiment are the best choice. In this situation, changing the G_ACT update frequency will also change the performance of the model. Figure 8 shows the relation between the G_ACT update frequency and the system performance. In these experiments, the strategies that are used are all the same as described in the second experiment of Table 2. The only difference is that the G_ACTs in the agents are updated with different frequencies, which may lead to differences in the amount of system workload for service advertisement. The best trade-off between discovery speed and system efficiency is once every 20 simulation steps in this example model.

In fact, the performance of this example model can be improved further if using agent level modelling. Different agents can use a mixture of different strategies to achieve higher performance of the whole system. These are not discussed in detail here.

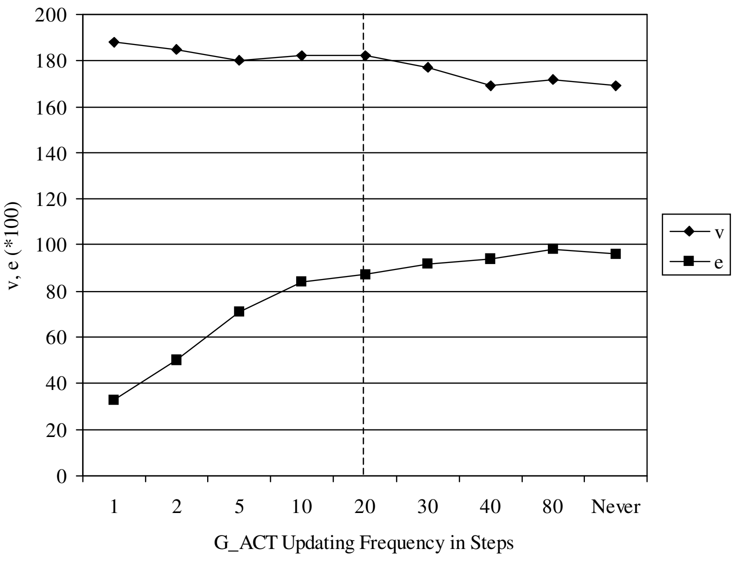


Fig. 8. Choice of G_ACT update frequency.

5. Conclusions

There will be many large-scale distributed software systems with high dynamics in future. In this work, an agent-based hierarchical model is developed to meet the requirements of the scalability. The system functions are also abstracted into the processes of the service advertisement and discovery at a high level. The performance issues arise from the high dynamics of the system. There are several common performance optimisation strategies that can be used in this system. However, the evaluation of their impacts on the system performance is difficult. In this work a modelling and simulation environment has been implemented to aid the performance evaluation process. Two case studies are given and the simulation results show the impact of the agent mobility and the choice of optimisation strategies on the overall system performance.

One of the ongoing works is adding hardware-level configuration for agent communications in the performance model. A reasonable estimation about the total service discovery time will be more useful to characterise the actual system than the number of the communications. Other work includes coupling the performance modelling and simulation of the agent system with the real implementation of agent-based service advertisement and discovery. Retrieving statistical data on services and requests in the system, real-time performance evaluation results can be produced to adjust performance optimisation strategies of the agents to improve the overall system performance.

References

1. K. Amold, B. O'Sullivan, R. Scheifer, J. Waldo and A. Woolrath, The Jini™ Specification, Addison Wesley, 1999.
2. J. P. Bigus and J. Bigus, Constructing Intelligent Agents with Java: A Programmer's Guide to Smarter Applications, Wiley Computer Publishing, 1998.
3. J. M. Bradshaw (ed.), Software Agents, The AAAI Press/The MIT Press, 1997.
4. J. Bray and C. Sturman, Bluetooth: Connect Without Cables, Prentice Hall, 2000.
5. J. Cao, Y. Fan and C. Wu, "Research and Design of Operational Administration Agents", Computer Integrated Manufacturing Systems - CIMS, 5(3), 1999, 39-43.
6. J. Cao, D. J. Kerbyson and G. R. Nudd, "Dynamic Application Integration Using Agent-Based Operational Administration", Proc. of 5th Int. Conf. on Practical Application of Intelligent Agents and Multi-Agent Technology, Manchester, UK, 2000, 393-396.
7. J. Cao, D. J. Kerbyson, E. Papaefstathiou and G. R. Nudd, "Performance Modelling of Parallel and Distributed Computing Using PACE", Proc. of 19th IEEE Int. Performance, Computing and Communication Conf., Phoenix, USA, 2000, 485-492.
8. J. Cao, D. J. Kerbyson and G. R. Nudd, "Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing", Proc. of 1st IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, 2001, 311-318.

9. J. Cao, D. J. Kerbyson and G. R. Nudd, "Use of Agent-Based Service Discovery for Resource Management in Metacomputing Environment", Proc. of 7th Euro. Conf. on Parallel Computing, Manchester, UK, 2001 (to appear).

10. D. Chauhan and A. D. Baker, "JAFMAS: A Multiagent Application Development System", Proc. of 2nd Int. Conf. on Autonomous Agents, Minneapolis/St. Paul, 1998, 100-107.

11. P. Ciancarini, "Coordination Models and Languages as Software Integrators", ACM Computing Surveys, 28(2), 1996, 300-302.

12. P. Ciancarini and A. L. Wolf (Eds.), Coordination Languages and Models – Proceedings of COORDINATION 1999, Lecture Notes in Computer Science 1594, Springer, 1999.

13. R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng and I. Soboroff, "Agent Development with Jackal", Proc. of 3rd Int. Conf. on Autonomous Agents, Seattle, Washington, U.S.A., 1999, 358-359.

14. M. Cremonini, A. Omicini and F. Zambonelli, "Modelling Network Topology and Mobile Agent Interaction: An Integrated Framework", Proc. of 1999 ACM Symposium on Applied Computing, The Menger, San Antonio, Texas, U.S.A., 1999, 410-412.

15. Y. Fan, W. Shi and C. Wu, "Enterprise Wide Application Integration Platform for CIMS Implementation", J. of Intelligent Manufacturing, 10(6), 1999, 587-601.

16. I. Foster, and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.

17. Y. Y. Goland, T. Cai, P. Leach, Y. Gu and S. Albright, "Simple Service Discovery Protocol/1.0: Operating without an Arbiter", IETF Internet Draft, 1999.

18. E. Guttman, C. Perkins, J. Veizades and M. Day, "Service Location Protocol, Version 2", RFC 2608, IETF Draft Standard, 1998.

19. N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers and J. L. Alty, "Implementing a Business Process Management System using ADEPT: A Real-World Case Study", Int. J. of Applied Artificial Intelligence, 14(5), 2000, 421-465.

20. N. R. Jennings and M. J. Wooldridge (eds), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

21. Jini, "Jini™ Architectural Overview", Sun Technical White Paper, Jan. 1999.

22. R. Lea, S. Gibbs, R. Gauba and R. Balaraman, HAVi Example By Example: Java Programming for Home Entertainment Devices, Prentice Hall, 2001.

23. T. W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination", ACM Computing Survey, 26(1), 1994, 87-119.

24. B. Miller and R. Pascoe, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer", Bluetooth White Paper, July 1999.

25. D. B. Moran, A. J. Cheyer, L. E. Julia, D. L. Martin and S. Park, "Multimodal User Interfaces in the Open Agent Architecture", Proc. of 1997 Int. Conf. on Intelligent User Interfaces, 1997, 61-68.

26. G. Papadopoulos and F. Arbab, "Coordination Models and Languages", in Advances in Computers, 46: The Engineering of Large Systems, Academic Press, 1998, 329-400.

27. R. Pascoe, "Building Networks on the Fly", IEEE Spectrum, 38(3), 2001, 61-65.

28. G. G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation", IEEE Internet Computing, 4(5), 2000, 18-26.

29. A. Sloman and B. Logan, "Building Cognitively Rich Agents Using the SIM_Agent Toolkit", Communications of the ACM, 42(3), 1999, 71-77.

30. UPnP, "Understanding Universal Plug and Play", Microsoft White Paper, Jun. 2000.