ELSEVIER

Data & Knowledge Engineering 41 (2002) 229–245

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Supporting Internet-scale multi-agent systems

N.J.E. Wijngaards [*], B.J. Overeinder, M. van Steen, F.M.T. Brazier

Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Received 14 December 2001; received in revised form 19 December 2001; accepted 19 December 2001

Abstract

The Internet provides a large-scale environment for (intelligent) software agents. Agents are autonomous (mobile) processes, capable of communication with other agents, interaction with the world, and adaptation to changes in their environment. Current approaches to support agents are not geared for large-scale settings. The near future holds thousands of agents, hosts, messages, and migratory movements of agents. These large-scale aspects require a new approach to facilitate the development of agent applications and support. AgentScape is a scalable agent-based distributed system, described in this paper, that aims at tackling these aspects. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Large-scale agent systems; Middleware; Scalability; Mobility; Heterogeneity; Interoperability

1. Introduction

Multi-agent systems are subject to much research with a focus mainly on small-scale homogeneous systems that are developed for specific domains of applications. In the near future, however, large-scale agent systems will emerge with vast numbers of agents that communicate, manipulate objects, and move across machines. Examples of large-scale agent systems include e-business applications (virtual shopping malls and auctions), Internet-wide data warehouses, and navigation systems. Unfortunately, current systems cannot simply be expanded to cover large heterogeneous networks such as the Internet, be managed in a distributed fashion by several administrative organizations at the same time, and support vast numbers of agents.

[*] Corresponding author. Tel.: +31-20-444-7756.
E-mail address: niek@cs.vu.nl (N.J.E. Wijngaards).

When considering scalability, there are a number of key aspects in agent systems that need further attention. First of all, what exactly is an agent is important. Agents form the active entities and, from a scalability perspective, should be designed to operate in very large systems. Related to this issue is the way interaction between agents takes place, that is, the communication and co-ordination among agents. Yet another key aspect is the ability of an agent to adapt to changes in its environment. In large-scale systems, an obvious assumption is that the environment changes continuously, imposing specific requirements on agents.

When dealing with an agent system that is geographically dispersed across a wide-area network, it is also important to take a look at whether and how agents migrate between locations. An important assumption is that these systems are not closed. Instead, they need to operate in an open (and thus extendible) way where agents from other agent platforms can arrive and leave, leading to security considerations. Finally, to support the development of agent systems facilitation, tools and platforms to host agents and services for agents, are important as well.

In this paper, agent systems are described from the perspective of two research communities: Artificial Intelligence (AI) and Computer Systems (CS). Subsequently, an overview is given of requirements for large-scale agent systems, and more details on scalability issues. Furthermore an outline of AgentScape is presented; a system explicitly designed to support large-scale agent systems.

The paper is organized as follows. In Section 2 a brief overview is given of current research on agents. Aspects of large-scale agent systems are described in Section 3. Current research on a scalable agent operating system, AgentScape, is described in Section 4. In Section 5 development of large-scale agent systems is discussed and suggestions for future research are presented.

2. Current agent systems

Agents and agent systems have been studied for over a decade, resulting in many papers from the AI community as well as from CS researchers. Research from the perspective of AI focuses on different aspects than CS-based research. This section describes current research on agent systems from both disciplines.

2.1. Agents from an AI perspective

From the AI perspective agents are (1) autonomous and pro-active, (2) may be mobile, (3) are capable of communication with other agents, (4) are capable of interaction with the outside "world," and (5) are most often intelligent (meaning that they may be capable of learning, have knowledge, can perform complex tasks, and can reason about and with this knowledge). In this respect, a well-known example of an agent is a human being. An agent (either human or automated) has its own environment, consisting of other agents and a (material) world. The agent metaphor offers a means to model situations with distributive activity on a conceptual level (e.g., [26]), and is related to research on Distributed Artificial Intelligence.

Different notions of agency have been proposed (e.g., [4,40,60,50]). The characteristics of *weak agency* defined by Wooldridge and Jennings [60] provide a means to reflect on the tasks an agent needs to be able to perform. Pro-activeness and autonomy are related to an agents ability to reason about its own processes, goals and plans. Reactivity and social ability are related to the ability to interact with the material world and to communicate with other agents. The ability to communicate and cooperate with other agents and to interact with the material world often relies on an agents ability to acquire and maintain its own knowledge of the world and other agents. In contrast, the notion of *strong agency* is based on the characteristics of mentalistic and intentional notions (related to the notion of intentional stance by Dennett [14]).

The model of weak agency is widely accepted within the agent community and forms a useful basis for a discussion on agent models. Most models have a degree of genericity: the agent model is not fully specified, and may be extended for agents in a specific application (e.g. the generic agent model [6]). A number of these specification languages are specifically developed for specifying BDI-agents, which adhere to the notion of strong agency.

The BDI *architecture* [46] is organised around the notions of Beliefs, Desires and Intentions. The beliefs are concerned with information on the environment (the world and other agents). Desires and intentions are concerned with the goals and plans of an agent. Specific (formal) agent model have been developed for BDI-agents [47,34], among which a refinement of the generic agent model for BDI-agents [5].

Agents typically interact with other agents. Agent communication languages (ACLs) and co-operation models are actively being researched.

An ACL describes "speech acts" in the form of message exchanges. KQML was a first attempt to come to a standardized ACL [16]. More recently, FIPA [11] is providing standards for ACLs and protocols [17]. Interesting is that the semantics of a message in an ACL is generally left open to the extent that it is associated with an ontology. For example, in the Semantic Web [1] effort [3] a rich ontology language is developed by which machines may understand information.

The many different ACLs and ontologies leads to an interoperability problem [61]. Solutions to this problem include the use of intermediary agents that find agents capable of translating between ontologies. Another approach is to devise information brokering agents that essentially act as intermediaries or gateways [32,31,53]. Mediating agents have also been the object of study in the area of security and privacy [58], where they are called "Alter-ego's", to emphasize their role as intelligent carriers of private information of human beings.

A completely different approach to interoperability is to replace message exchange by specific coordination models. Coordination models may be used to set up and maintain aggregates of cooperating agents. The purpose of a coordination model is to enable the integration of a number of agents in such a way that the resulting ensemble executes as a whole. However, the temporal and referential coupling between agents in message-based systems is replaced by an associative memory that decouples communicating parties in time and space [42].

As agent systems grow in complexity the density and diversity of the interconnections between agents increase rapidly. The global behaviour of these complex systems can be considered to be an emergent property of the interactions between the different agents. An alternative way to manage this form of agent-based system is to utilize emergent properties to make self-organizing and self-regulating multi-agent systems [24].

[1] See also the article by Klein et al. in this special issue on the Semantic Web.

Agents operate in a dynamic environment, making it impossible to statically determine an agent's optimal behaviour in advance. Agents have to learn from and adapt to their environment. Multi-agent learning and adaptation, that is, the ability of agents to learn how to cooperate and compete, become crucial [29]. Adaptive and learning agents are often employed to maintain profiles of (human) agents. For example, a consumer agent maintains a profile of the personal assistant agent, and adapts this profile on the basis of interaction with the personal assistant agent (e.g. as also encountered in negotiation settings [10,12]). Adaptation may entail mobility of an agent: an agent may migrate to a specific "place" to access locally available resources.

Agents, as studied within AI, are (to some extent) intelligent, and interact to solve complex problems. The overall behaviour of multi-agent systems is an important topic of current research. However in AI, the implementation of agents is not of great importance, usually only for demonstration purposes. Efficiency and performance is not usually considered to be important.

2.2. Agents from a CS perspective

An agent from a CS perspective is often considered to be just a process [54], a piece of running code with data and state. The functionality of these agents can most often be described in terms of human behaviour, and to which the predicate intelligent is associated. Agents are processes that are autonomous and pro-active (capable of making "their own" decisions when they like), interacting, and may be mobile.

Agents are often related to objects, where the latter are generally considered to be passive [27]. In other words, an object needs to be invoked in order to perform a function, and performs only during an invocation. Agents, on the other hand, receive messages and autonomously decide if, when, and how to (re)act. The only way to influence an agent is by sending requests; the agent stays in complete control and may perform functions even if not requested to do so.

Agents usually interact by exchanging messages, where message delivery is subject to different "qualities of service." For example, the message paradigm described by FIPA prescribes reliable and ordered point-to-point communication between agents [18].

Agents in CS are often mobile. The decision to migrate is taken autonomously by the mobile agent itself. The ability of migration provides mobile agents a means to overcome the high latency or limited bandwidth problem of traditional client–server interactions by moving their computations to required resources or services. The current evolution of intelligent and active networks in system and network management, for example, is based on this technology. A similar tendency is observed in the search and filtering of globally available information such as in the electronic marketplaces, e-commerce, and information retrieval on the World Wide Web [23].

A distinction can be drawn based on whether the execution state is migrated along with the unit of computation or not [20]. Systems providing the former option are said to support *strong mobility*, as opposed to systems that discard the execution state across migration, and are hence said to provide *weak mobility*. In systems supporting strong mobility, migration is completely transparent to the migrated program, whereas with weak mobility, extra programming is required in order to save part of the execution state.

Strong mobility as found in NOMADS [51], Ara [44], and D'Agents [22], requires that the entire state of the agent, including its execution stack and program counter [25], is saved before the agent is migrated to its new location. Despite the advantages of strong mobility, many agent systems support weak mobility (like Ajanta [56] and Aglets [30]). Most of the agent systems are implemented on top of the Java Virtual Machine (JVM), which provides with object serialization basic mechanisms to implement weak mobility. The JVM does not provide mechanisms to deal with the execution state.

Security is of great importance in agent systems, not only in electronic monetary transactions, but also that mobile agents should not become the next generation of viruses. Current research on secure agent systems concentrates mainly on protecting hosts against hostile mobile agents. The problem of security stems from the fact that untrusted code needs to be executed. Modern solutions are based on the notion of protection domains by which a security policy for accessing local resources can be enforced [21]. Only very few systems also provide facilities for protecting mobile agents against hostile hosts [28].

2.3. Agent systems

Agents are used in a wide variety of applications [27]: process control, manufacturing, air traffic control, information management, e-commerce, business process management, patient monitoring, health care, games, and interactive theater and cinema. Deploying an agent system (or prototype thereof) includes not only modelling autonomous, interactive, intelligent agents, but also actually implementing these agents as well as an underlying infrastructure.

Agents can exist only by virtue of some kind of *agent platform*. Such a platform runs on a relatively small collection of machines and provides basic facilities such as creating and running an agent, searching for an agent, migrating agents to other platforms, and enabling the basic communication with other platforms that host an agent. Platforms and the facilities for modelling agents integrate the AI and CS perspective of agents, leading to what we refer to as agent systems.

Facilitation of (intelligent) agents commonly includes tools to model and develop agents, and an agent platform. Most agent frameworks provide some tools for modelling and developing agents, ranging from code libraries to interactive agent-design tools, with which AI specifications of agents are transformed into executable descriptions of agents.

It can be argued that only relatively few agent systems exist that integrate the AI and CS perspective as discussed above. ZEUS [39], NOMADS [51], Sensible Agents [2] and Ajanta [56] are approaches to agent systems in which tools for developing agents are combined with agent platforms. Basic facilities such as communication, creation and deletion of agents, as well as mobility are provided by most agent platforms. Aspects such as interoperability, efficiency and performance, but also security, are part of the current research.

3. Large-scale agent systems

Large-scale agent systems currently do not exist. A number of agent-based simulation systems involves larger numbers of agents, e.g. [37]. Often in agent-based simulation systems, the agents involved are kept small and/or simple (to facilitate simulation). However, with the advent of the Internet, large-scale agent systems will become more prevalent.

Ideally, multi-agent systems are highly dynamic open systems, with an ever-changing population of agents: new agents emerge (or are created), existing agents die, move, learn/forget etc.

The dynamics of such systems are hard to predict when considering the number of messages sent, migrationary movements of agents, birth and death of agents, etc. The number of agents in large-scale distributed applications such as e-business applications (virtual shopping malls and auctions), Internet-wide data warehouses, and navigation systems, can vary considerably over time. The systems need to be able to scale (in terms of the number of agents and available resources) almost immediately without noticeable loss of performance, or considerable increase in administrative complexity [38].

This problem of scalability is not an AI problem in itself. It is a problem with which the distributed computing community is still wrestling. A solution requires collaboration between these two disciplines. In the next section the topic of scalability in large-scale agent systems is addressed from the agents' perspective and the systems' perspective [8].

3.1. Agent-level scalability

Scalability is an important, yet under-researched, aspect of agent platforms. A number of multi-agent frameworks and environments have been developed to construct multi-agent systems, but not for systems with (very) large numbers of agents. Not only does the number of agents increase, but also the number of agent interactions, movements of agents, agents inserted into the system, agents deleted from the system, etc. All of these issues require a scalable solution; solutions that may conflict.

One aspect of current research on multi-agent systems is that a large system is deemed to consist of hundreds of agents, maybe a thousand, but not millions. The claim that Auctionbot is scalable, for example, is supported by an experiment with only 90 agents [62]. Larger numbers of agents require scalable development frameworks and support environments.

The term "scalability" is not always used to refer to architecture, services and performance. In some cases it is used to refer to scalable functionality. For example, the SAIRE approach [41] claims to be scalable because it supports heterogeneous agents. Shopbot [15] claims to be scalable because its agents can adapt to understand new websites. In both cases, the term *extensible functionality* would seem to be more appropriate.

Researchers and developers of multi-agent frameworks are beginning to realize that scalability is an issue. Some multi-agent frameworks rely on another framework to solve the problem of scalability. For example, scalability in the CoABS (DARPA Control of Agent Based Systems) approach [55] is based on adequate support from computational grids in providing a plug-in infrastructure for agents [19].

In other multi-agent frameworks, aspects of scalability are specifically addressed. In ZEUS [13] scalability is defined to be the growth rate of the maximum communication load, that is, the number of messages sent across a single link grows at worst linearly with the number of agents. This addresses a loss of performance problem, and is a step towards developing scalable multi-agent frameworks. In open agent architecture (OAA) [33] matchmaking agents are described that can handle larger number of agents. The RETSINA MAS infrastructure [52] is designed to support multi-agent systems that run on a number of LANs and to avoid single-point of failures (e.g. in agent name services).

Turner and Jennings [57] propose to (automatically) change the organisation of agents in the multi-agent system as a response to an increase in the population of a multi-agent system. For example, more middle agents or matchmakers are introduced to reduce overhead. Their approach is a possible step towards addressing administrative problems related to scalability.

None of the aforementioned approaches addresses minimizing the loss of performance as well as minimizing administrative overhead. Research on fully automated creation and adaptation of agents, open (extensible) agent systems, or large-scale agent life-cycle management, has yet to begin.

Research on specific services in multi-agent systems such as directory services also addresses scalability. The approach taken by Shehory [49] is an example of a theoretical analysis in which agents locate agents based on each agent's own caching lists of agents they know. No experiments have yet been conducted.

3.2. Systems-level scalability

A varying number of (heterogeneous) platforms may require some form of automated linking (and unlinking) of the platform to a large-scale agent system. Platforms, of course, differ extensively in the facilitation of the development of agents, and the support and functionality offered for agents and objects. Machines may be added or removed and machines may differ in the availability of resources (consider, e.g., Unix and Windows NT systems versus hand-held devices).

A platform provides services for agents. Location services are used to obtain the address of an agent (or object) on the basis of the name of the agent (or identifier of the object). Directory services are used for attribute-based searching and matching by which names of agents (or identifiers of objects) are acquired. An increase in the number of agents and objects plus the number of updates requires scalable solutions [1].

Agents need to be managed with respect to their creation, deletion, and migration. When needed, agents may be created by other agents, and become part of the agent system. The creation service may clone an agent or create an agent on the basis of an available description. Similarly, an object-creation service is needed. The number of agents migrating from machine to machine (and possibly across platforms) is greatly increased.

Within a large-scale agent system, agents are heterogeneous, and so are platforms supporting agents. Migration of an agent involves migrating its code (and data) from one platform to another. Four migration scenarios of agents can be distinguished [7]. (1) Homogeneous migration is when an agent migrates to another host without any changes to the format of its executable code or its interfaces to the agent platform. (2) Cross-platform migration is when an agent migrates to another host with a different agent platform, but that offers the same (virtual) machine interface. (3) Agent-regeneration migration is when an agent migrates to a host running a different (virtual) machine requiring that the agent is regenerated, resulting in different executable code. (4) Heterogeneous migration is when an agent migrates to another host with a different agent platform and offering a different (virtual) machine. In this case, regeneration of the agent is necessary. The migration scenarios in which the agent is regenerated may be realized by using an agent factory [9]. Instead of migrating the "code" (including data and state) of an agent, a blueprint of an agent's functionality and its state is transferred. An agent factory generates new code on the basis of this blueprint.

From the systems point of view, scalability problems generally manifest themselves as performance problems. Three scaling techniques are discussed which may be used to minimize loss of performance: (1) hiding communication latencies, (2) distribution, and (3) replication.

Hiding communication latencies is applicable in the case of geographical scalability, that is, when an agent system needs to span a wide-area network. To avoid waiting for responses to requests that have been issued to remote agents or services, the requesting agent is programmed to do other useful work. This approach does require that an agent can be interrupted when the expected response (if any) is to be delivered.

Distribution generally involves partitioning a (large) set of data into parts that can be handled by separate servers. A well-known example of distribution is the natural partitioning of the set of Web pages across the approximately 25 million Web servers that are currently connected through the Internet. Other examples of distribution include the vertical or horizontal partitioning of tables in distributed databases [43].

When considering large-scale networks like the Internet it becomes crucial to combine distribution with latency hiding. Unfortunately, this is not always possible, for example when an agent simply needs an immediate response.

A third, and widely-applied technique is to place multiple copies of data sets across a network, also referred to as *replication*. The underlying idea is that by placing data close to where they are used, communication latency is no longer an issue, so that agent-perceived performance is high. Having multiple copies means that such performance is good for all agents, no matter where they are located.

Unfortunately, replication introduces a serious problem. Whenever a replica is updated, that replica becomes inconsistent with the other replicas. Matters become worse when multiple concurrent updates need to be carried out simultaneously, because all replicas have to be the same after all updates have been processed. Keeping replicas consistent introduces a consistency problem that can be solved only by means of global synchronization. However, global synchronization in a large-scale network is inherently nonscalable, as it requires communication between *all* parties that are to be synchronized.

The only solution to the consistency problem is to allow replicas to be somewhat out of synch with respect to updates. In other words, a weak consistency model is adopted. The form of, and to what extent weak consistency can be tolerated is highly application dependent. As a consequence, scalable multi-agent systems will need to support configurable and perhaps even adaptive replication strategies. No single strategy will show to be optimal under all conditions. Even for relatively simple systems such as the Web, differentiating strategies can make a lot of difference [45].

Scalability in large-scale agent systems is a difficult and challenging problem. Models for security in large-scale agent systems are not common yet. In fact, security by itself introduces inherent scalability problems as we often need to set up secure point-to-point channels, introducing referential and temporal coupling of agents. When dealing with security within groups, providing scalable solutions turns out to be even harder (see e.g. [48]).

4. AgentScape

The aspects of large-scale agent-based systems as described in the previous section, form the basis for the design goals of the AgentScape agent platform. The rationale behind the design decisions are (1) to provide a platform for large-scale agent systems, (2) support multiple code bases and operating systems, and (3) interoperability with other agent platforms. The consequences of the design rationale with respect to agents and objects, interaction, mobility, security and authorization, and services are presented in the next sections.

4.1. Approach

AgentScape is a middleware layer that supports large-scale agent systems. The overall design philosophy is "less is more," that is, the AgentScape middleware should provide minimal but sufficient support for agent applications, and "one size does not fit all," that is, the middleware should be adaptive or reconfigurable such that it can be tailored to a specific application (class) or operating system/hardware platform.

Agents and objects are basic entities in AgentScape. Thus any multi-agent application in AgentScape is composed of agents and objects. Agents are active entities in AgentScape that interact with each other by message-passing communication. Furthermore, agent migration in the form of weak mobility is supported (see Section 2.2). Objects are passive entities that are only engaged into computations reactively on an agent's initiative.

Basic services in AgentScape are naming and location services for agents, objects, and locations. These services enable agents to find and contact other agents or objects in the distributed multi-agent system, and to migrate to other locations.

Scalability, heterogeneity, and interoperability are important principles underlying the design of AgentScape. The design of AgentScape includes the design of agents and objects, interactions, migrations, security and authorization, as well as the agent platform itself. For example, scalability of agents and objects is realized by distributing objects according to a per-object distribution strategy, but not agents. Instead, agents have a public representation that may be distributed if necessary.

4.2. AgentScape model

The design goal of AgentScape is to define a minimal set of functionality and services that should be supported by the middleware of a scalable agent-based distributed system. However, AgentScape must be extensible and useable as a kernel for other agent platforms. For development of agent applications, an application programming interface (API) and a runtime system (RTS) are provided. However, the default API and RTS can be extended to provide a higher-level application programming interface with, for example, a model that offers more structure and semantics to the agent application developer.

4.2.1. AgentScape model from an AI perspective

Central entities in AgentScape are agents and objects. An agent may communicate with other agents and interact with objects. All inter-agent or agent–object interaction is realized via interfaces that are bound to the calling (or interaction initiating) agent. Agents and objects reside at a location, which is an abstraction from a physical location. A location contains a set of resources: host(s), disks, network, main memory, etc. For a consistent model, a location comprising the AgentScape middleware and resources presents itself as an agent and a set of objects. The AgentScape middleware is represented by a *location manager* agent, and the resources by their respective objects. Hence, the AgentScape middleware interface presents itself as a normal agent

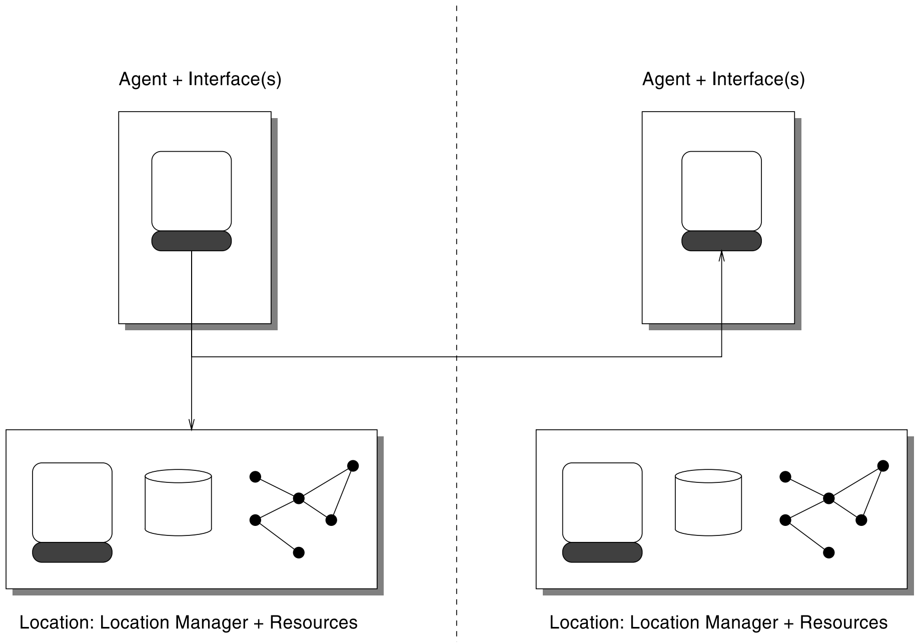
Agent + Interface(s)
Agent + Interface(s)
Location: Location Manager + Resources
Location: Location Manager + Resources

Fig. 1. The AgentScape model from the perspective of agents.

interface on which operations or methods can be called upon. The difference between the location manager and an arbitrary other agent is the authority of the location manager when it comes to matters such as creating, destroying, and moving agents and objects.

Fig. 1 presents a model of AgentScape from the agent perspective, that is, the location comprising the middleware and the resources are represented by a location manager agent and resource objects. Calls from an agent to the middleware are modeled by requests to the location manager agent to, for example, create an agent or move an agent. Information about resources residing at the location can be retrieved by binding to the resource objects, which are *local* distributed objects. These objects can be accessed only within the location they reside, not from outside the location.

The terms location manager and middleware are used interchangeably, depending on the context. We use the term location manager for the agent representation of the middleware towards the agents, and the term middleware in the context of system architecture.

4.2.2. AgentScape model from a CS perspective

The basic idea in the AgentScape model is that most of the functionality is provided by the agent interface implementations such that the middleware (or the agent representation of the middleware) can be designed to perform basic functions. This approach has a number of advantages. First as the middleware must provide basic functionality, the complexity of the design of the middleware can be kept manageable and qualities like robustness and security of the middleware can be more easily asserted. Additional functionality can be implemented in the agent-specific interface implementation. Fig. 2 depicts a model of the agent-specific interface and the

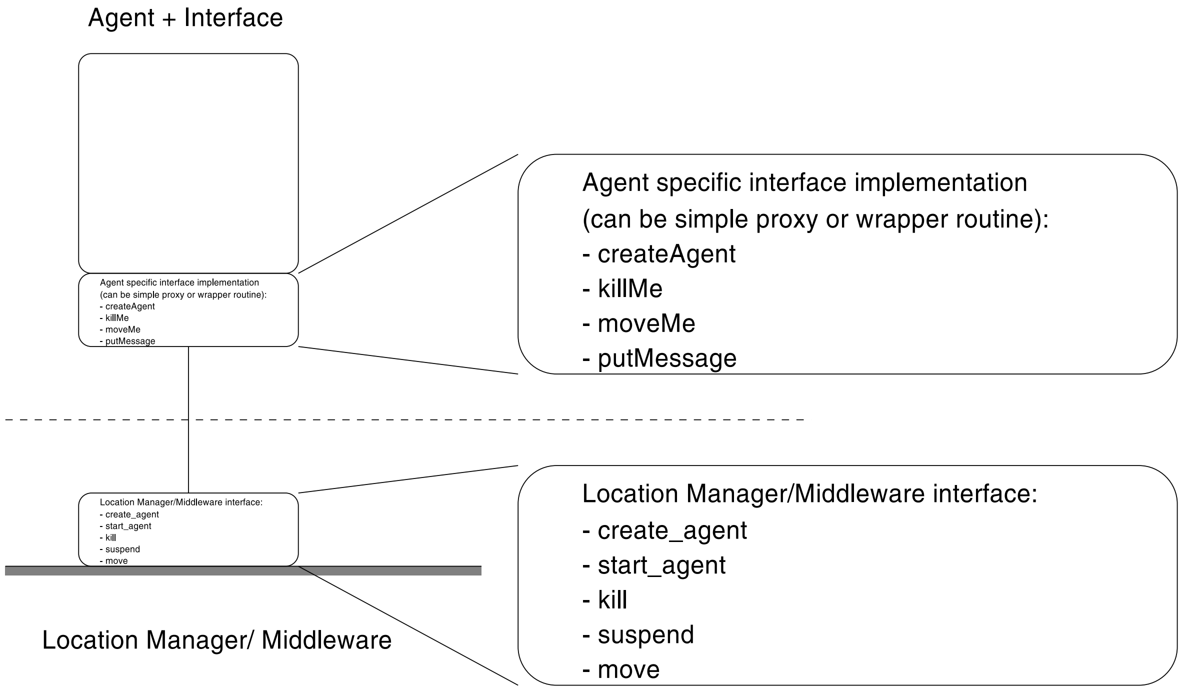
Agent + Interface
Agent specific interface implementation
(can be simple proxy or wrapper routine):
- createAgent
- killMe
- moveMe
- putMessage
Location Manager/Middleware interface:
- create_agent
- start_agent
- kill
- suspend
- move
Location Manager/ Middleware

Fig. 2. The AgentScape model from the perspective of agents.

location-manager interface, and how agent interface calls can be broken down in more basic location-manager (middleware) calls.

For example, the interface call to migrate an agent can have different specific implementations of agent migration. With basic agent migration, the agent can be suspended and be in transit to another location for an arbitrary period of time. However, if there is a high availability requirement for the agent, another agent-migration interface implementation can be provided that realizes an "instantaneous" move of the agent to the new location (e.g. by first creating and starting the remote agent, then informing name and location servers of the new contact address, then start forwarding messages to the new agent, and finally ask the location manager to cleanup or kill the old agent).

Agent–agent interaction is exclusively via message-passing communication. Asynchronous message passing has good scalability characteristics with a minimum of synchronization between the agents. Tuple spaces also provide a mechanism for communication that does not enforce synchronization between the communicating partners, but cannot enforce the actual receipt of the information.

Agent migration between locations is based on weak mobility. The *state* of the agent is captured (e.g. the variables referenced by the agent) but not the *context* of the agent (e.g. stack pointer and program counter). Requests for migration are directed by message passing to the agent. Hence, the agent always receives requests for migration and has to agree to participate in the migration activities by serializing its state. The middleware also sends requests for migration to agents, as message passing is the only supported communication/interaction mechanism.

Global and local distributed objects in AgentScape are Globe objects [59]. Global distributed objects have their own replication strategy to distribute and replicate their internal state across multiple locations.

4.3. AgentScape experimental framework

AgentScape provides a framework for research on a number of AI and CS topics. The design and realization of an Internet-scale multi-agent platform should consider scalability as a leading design requirement. With respect to agents and objects, this implies the support of scalable methods for agent management, including creation and deletion of agents, migration, and agent name and location services [1]. With the dynamic creation of many agents, effective support is necessary for keeping track of the agents during their lifespan. For example, during a certain phase of the computation, such as when new results become available, distributed mobile agents may need to be recalled and may have to either migrate back to the originating location or may need to be destroyed.

In large-scale systems, latency hiding and loose synchronization are important qualities of agent interaction. Unnecessary blocking and synchronization between distant agents that communicate over wide-area Internet connections, nullifies the performance of any multi-agent system. Hence, support for asynchronous (non-blocking) communication mechanism is essential, although in some situations synchronous interaction is desirable and should be included in the model (e.g. for service requests to the middleware like name lookup).

Scalable security services can be based on public key infrastructures and authorization delegation (by proxy). Authentication of agents and objects can be verified by the signatures that are attached to them. With public-key infrastructures this authentication can be realized locally (once the public key of the entity is locally available). Delegation of authorization enables other entities (agents) to act on your behalf such that there is no central entity where all requests have to be directed to.

Within AgentScape, management of large-scale agent systems is an important issue, including not only life-cycle management of agents, but also management of security and authentication, middleware configuration, and resources. As centralized management mechanisms are not applicable (scalability), other approaches need to be considered such as law-governed coordination mechanisms [36].

The code base and operating system independence requirement implies that agents can be, for example, Java, Python, or C programs, and run on, e.g., Unix or Windows NT operating systems. The consequence is that the runtime support for the agents must hide the details of the underlying language and operating system details. The middleware already abstracts from many aspects of the operating system, for example, it supports the creation and deletion of agents and objects. Language specific aspects are hidden by the runtime support, for example, the runtime support solves the peculiarities of creating a communication channel with the middleware. Another consequence of code base and operating system independence is that the system cannot rely on, for example, the Java security infrastructure. The security architecture should be platform independent. Furthermore, the security architecture must be an integral part of the design of the AgentScape agent platform, and should not be regarded as an add-on and of a later concern. With respect to mobility of agents, the migration of agents (suspend, migration, and restart) must be language independent from the middleware perspective. Thus from the middleware concern, an image of an agent is transported to another location, and at the new location it is handed over to an appropriate interpreter to recreate and restart the agent.

Interoperability between agent platforms can be realized in two ways. First by conforming to standards like FIPA [11] or OMG MASIF [35]. These agent platform standards define interfaces and protocols for interoperability between different agent platform implementations. For example, the OMG MASIF standard defines agent management, agent tracking (naming services), and agent transport amongst others. The FIPA standard is more comprehensive in that it defines also agent communication and agent message transport, and even defines an abstract architecture of the agent platform. A second approach to interoperability is realized by reconfiguration or adaptation of the mobile agent. This can be accomplished by an agent factory as described in Section 3.2 (see also [7]), which regenerates an agent given a blueprint of the agent's functionality and its state, using the appropriate components for interoperability with the other agent platform.

5. Discussion

Multi-agent systems are becoming more prevalent. Although current research on agent systems is focused on small-scale agent systems, soon vast numbers of agents will be deployed in large-scale agent systems. The basic infrastructure is already available: the Internet is a large heterogeneous network managed in a distributed fashion. Support for large-scale agent systems is not present yet. Large-scale agent systems pose additional requirements on agents and support for agents.

The main aspects of large-scale agent systems are their extendibility, heterogeneity, interoperability, and scalability. A large-scale agent system is a highly dynamic system, in which the numbers and availability of agents, languages, ontologies, etc. may vary over time. The entities present in a large-scale agent system are very different from one another, leading to interoperability issues among agents and migrationary problems. As not only the number of entities but also the number of (inter)actions by agents is on a larger scale in large-scale agent systems, scalability is of paramount importance.

AgentScape addresses some important research issues regarding the realization of large-scale agent systems, such as scalability, language/platform independence, and interoperability. From the AI point of view, AgentScape is an extensible framework for the development of large-scale agent systems, such as more structured models for the environment of agents. From the CS point of view, AgentScape provides an experimental framework for studying (validation and verification) of new approaches in large-scale agent systems, such as scalable directory services, security, and interaction and coordination models.

The AgentScape model presented effectively supports both AI and CS research interests. On the AI side: extensible interfaces can be easily adopted to provide new development models for agent systems. On the CS side: the integrated modular approach, where platform dependent and independent parts are separated, and new functionality can be easily integrated or replaced with different designs makes AgentScape a flexible framework.

Supporting large-scale agent systems involves solving numerous problems on both the agent-level and the system-level. In this research area, much progress may be achieved by cooperation between researchers from both the AI and CS communities.

Acknowledgements

This work was supported in part by NLnet Foundation, http://www.nlnet.nl/. Discussions on the use of AgentScape with Andrew Tanenbaum and Guido van't Noordende were fruitful and led to a number of insights.

References

[1] G. Ballintijn, M. van Steen, A.S. Tanenbaum, Scalable user-friendly resource names, IEEE Internet Computing 5 (5) (2001) 20–27.
[2] K.S. Barber, R. McKay, A. Goel, D. Han, J. Kim, T.H. Liu, C.E. Martin, Sensible agents: the distributed architecture and testbed, IEICE Transactions on Communications, vol. 5, 2000, pp. 951–960, IECIA/IEEE Joint Special Issue on Autonomous Decentralized Systems, E83-B.
[3] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 5 (2001).
[4] J.M. Bradshaw (Ed.), Software Agents, AAAI Press/MIT Press, Menlo Park, CA, 1997.
[5] F.M.T. Brazier, B.M. Dunin-Keplicz, J. Treur, L.C. Verbrugge, Modelling internal dynamic behaviour of BDI agents, in: J.-J.C. Meyer, P.Y. Schobbes (Eds.), Formal Models of Agents (Selected papers from final ModelAge Workshop) vol. 1760 of Lecture Notes in AI, Springer Verlag, 1999, pp. 36–56.
[6] F.M.T. Brazier, C.M. Jonker, J. Treur, Compositional design and reuse of a generic agent model, Applied Artificial Intelligence 14 (2000) 491–538.
[7] F.M.T. Brazier, B.J. Overeinder, M. van Steen, N.J.E. Wijngaards, Agent factory: generative migration of mobile agents in heterogeneous environments, in: Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002), Madrid, Spain, 2002, pp. 101–106.
[8] F.M.T. Brazier, M. van Steen, N.J.E. Wijngaards, On MAS scalability, in: T. Wagner, O. Rana (Eds.), Proceedings of Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, May 2001, pp. 121–126.
[9] F.M.T. Brazier, N.J.E. Wijngaards, Automated servicing of agents, AISB journal 1 (1) (2002) 5–20.
[10] H.H. Bui, D. Kieronska, S. Venkatesh, Learning other agents' preferences in multiagent negotiation, in: Proceedings of the National Conference on Artificial Intelligence (AAAI-96), 1996, pp. 114–119.
[11] J. Dale, E. Mamdani, Open standards for interoperating agent-based systems, Software Focus 2 (1) (2001) 1–8.
[12] M. Dastani, N. Jacobs, C.M. Jonker, J. Treur, Modeling user preferences and mediating agents in electronic commerce, in: F. Dignum, C. Sierra (Eds.), Agent-Mediated Electronic Commerce, vol. 1991 of Lecture Notes in AI, Springer Verlag, 2001, pp. 164–196.
[13] P. De Wilde, H.S. Nwana, L.C. Lee, Stability, fairness and scalability of multi-agent systems, International Journal of Knowledge-Based Intelligent Engineering Systems 3 (2) (1999) 84–91.
[14] D.C. Dennett, The Intentional Stance, MIT Press, Cambridge, MA, 1987.
[15] R.B. Doorenbos, O. Etzioni, D.S. Weld, A scalable comparison-shopping agent for the World Wide Web, in: Proceedings of the First International Conference on Autonomous Agents (Agents'97), Marina del Rey, CA, 1997, pp. 39–48.
[16] T. Finin, Y. Labrou, J. Mayfield, KQML as an agent communication language, in: J. Bradshaw (Ed.), Software Agents, MIT Press, Cambridge, 1997, pp. 291–316.
[17] FIPA. FIPA ACL message structure specification, 2000. http://www.fipa.org.
[18] FIPA. FIPA agent platform, 2001. http://www.fipa.org.
[19] I. Foster, C. Kesselman (Eds.), Computational Grids: The Future of High Performance Distributed Computing, Morgan Kaufman, San Mateo, CA, 1998.
[20] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Transactions on Software Engineering 24 (5) (1998) 342–361.
[21] L. Gong, R. Schemers, Implementing protection domains in the Java development Kit 1.2, in: Symposium Network and Distributed System Security, San Diego, CA, Internet Society, March 1998.

[22] R.S. Gray, G. Cybenko, D. Kotz, R.A. Peterson, D. Rus, D'Agents: applications and performance of a mobile-agent system. Software: Practice and Experience, 2002, in press.
[23] V.N. Gudivada, V.V. Raghavan, W.I. Grosky, R. Kasanagottu, Information retrieval on the World Wide Web, IEEE Internet Computing 1 (5) (1997) 58–68.
[24] J.H. Holland, Hidden Order: How Adaptation Builds Complexity, Perseus Books, Cambridge, MA, 1995.
[25] K.A. Iskra, F. van der Linden, Z.W. Hendrikse, B.J. Overeinder, G.D. van Albada, P.M.A. Sloot, The implementation of dynamite: an environment for migrating PVM tasks, Operating Systems Review 34 (3) (2000) 44–55.
[26] N.R. Jennings, On agent-based software engineering, Artificial Intelligence 117 (2) (2000) 277–296.
[27] N.R. Jennings, W.J. Wooldridge (Eds.), Agent Technology: Foundations Application and Markets, Springer-Verlag, Berlin, 1998.
[28] N. Karnik, A. Tripathi, Security in the ajanta mobile agent system, Software: Practice and Experience 31 (4) (2001) 301–329.
[29] D. Kudenko, E. Alonso, Learning in agents and multi-agent systems, Knowledge Engineering Review, 2002, in press.
[30] D.B. Lange, M. Oshima, G. Karjoth, K. Kosaka, Aglets: programming mobile agents in Java, in: Worldwide Computing and Its Applications, Lecture Notes in Computer Science, vol. 1274, Springer-Verlag, Berlin, 1997, pp. 253–266.
[31] A.Y. Levy, Y. Sagiv, D. Srivastava, Towards efficient information gathering agents, in: Software Agents, Proceedings of the AAAI 1994 Spring Symposium, 1994, pp. 64–70.
[32] P. Maes, Agents that reduce work and information overload, Communications of the ACM 37 (7) (1994) 31–40.
[33] D. Martin, A. Cheyer, D. Moran, The open agent architecture: a framework for building distributed software systems, Applied Artificial Intelligence 13 (1/2) (1999) 91–128.
[34] J.-J.C. Meyer, P.-Y. Schobbens, in: Formal Models of Agents, ESPRIT Project Model Age Final Workshop, Selected Papers, Springer Lecture Notes in AI, vol. 1760, Springer-Verlag, 1999.
[35] D. Milojicic et al., in: MASIF: the OMG mobile agent system interoperability facility, the Second International Workshop on Mobile Agents Lecture Notes in Computer Science, vol. 1477, Springer-Verlag, Berlin, September 1998, pp. 50–67.
[36] N. Minsky, V. Ungureanu, Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems, ACM Transactions on Software Engineering and Methodology (TOSEM) 9 (3) (July 2000) 273–305.
[37] S. Moss, Critical incident management: an empirically derived computational model, Artificial Societies and Social Simulation 1 (4), October 1998. http://www.soc.surrey.ac.uk/JASSS/1/4/1.html.
[38] B. Neuman, Scale in distributed systems, in: T. Casavant, M. Singhal (Eds.), Readings in Distributed Computing Systems, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 463–489.
[39] H. Nwana, D. Ndumu, L. Lyndon, J. Collis, ZEUS: a toolkit and approach for building distributed multi-agent systems, in: Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99), 1999, pp. 360–361.
[40] H.S. Nwana, Software agents: an overview, The Knowledge Engineering Review 11 (3) (1996) 205–244.
[41] J.B. Odubiyi, D.J. Kocur, S.M. Weinstein, N. Wakim, S. Srivastava, C. Gokey, J. Graham, SAIRE: a scalable agent-based information retrieval engine, in: Proceedings of the First International Conference on Autonomous Agents, Marina del Rey, CA, February 1997, pp. 292–299.
[42] A. Omicini, G.A. Papadopoulos, Coordination models and languages in AI, Applied Artificial Intelligence 15 (1) (2001) 11–103.
[43] T. Özsu, P. Valduriez, Principles of Distributed Database Systems, second ed., Prentice Hall, Upper Saddle River, NJ, 1999.
[44] H. Peine, T. Stolpmann, The architecture of the Ara platform for mobile agents, in: Proceedings of the First International Workshop on Mobile Agents (MA'97), Lecture Notes in Computer Science, vol. 1219, Springer-Verlag, Berlin Germany, April 1997, pp. 50–61.
[45] G. Pierre, I. Kuz, M. van Steen, A. Tanenbaum, Differentiated strategies for replicating Web documents, Computer Communications 24 (2) (2001) 232–240.

[46] A.S. Rao, M.P. Georgeff, Modeling rational agents within a BDI architecture, in: R. Fikes, E. Sandewall (Eds.), Proceedings of the Second International Conference on Knowledge Representation and Reasoning, Morgan Kaufman, 1991, pp. 473–484.
[47] A.S. Rao, M.P. Georgeff, BDI agents: from theory to practice, in: Proceedings of the First International Conference on Multiagent Systems, San Francisco, CA, 1995, pp. 312–319.
[48] M.K. Reiter, How to securely replicate services, ACM Transactions on Programming Languages and Systems 16 (3) (1994) 986–1009.
[49] O. Shehory, A scalable agent location mechanism, in: Intelligent Agents VI, Lecture Notes in Artificial Intelligence, vol. 1757, Springer-Verlag, Berlin, 1999, pp. 162–172.
[50] Y. Shoham, Agent-oriented programming, Artificial Intelligence 60 (1) (1993) 51–92.
[51] N. Suri, J.M. Bradshaw, M.R. Breedy, P.T. Groth, G.A. Hill, R. Jeffers, T.S. Mitrovich, B.R. Pouliot, D.S. Smith, Nomads: toward a strong and safe mobile agent system, in: Proceedings of the Fourth International Conference on Autonomous Agents, 2000, pp. 163–164.
[52] K. Sycara, M. Paolucci, M. van Velsen, J. Giampapa, The RETSINA MAS infrastructure, Autonomous Agents and Multi-Agent Systems, Journal of Autonomous Agents and Multi-Agent Systems, in press.
[53] K. Sycara, D. Zeng, Multi-agent integration of information gathering and decision support, in: Proceedings of the Twelvth European Conference on Artificial Intelligence (ECAI'96), 1996, pp. 549–553.
[54] A.S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, Prentice Hall, Upper Saddle River, New Jersey, 2002.
[55] C. Thompson, T. Bannon, P. Pazandak, V. Vasudevan, Agent for the masses, in: Proceedings of the Workshop on Agent based High Performance Computing: Problem Solving Applications and Practical Deployment, Third International Conference on Autonomous Agents Seatle, May 1999.
[56] A. Tripathi, N. Karnik, M. Vora, T. Ahmed, R. Singh, Mobile agent programming in Ajanta, in: Proceedings of the Ninteenth International Conference on Distributed Computing Systems (ICDCS'99), Austin, TX, May 1999, pp. 190–197.
[57] P.J. Turner, N.R. Jennings, Improving the scalability of multi-agent systems, in: Proceedings of the First International Workshop on Infrastructure for Scalable Multi-Agent Systems, Barcelone, Spain, June 2000.
[58] R. van de Riet, A. Junk, E. Gudes, Security in cyberspace: a knowledge-base approach, Data and Knowledge Engineering 24 (1) (1997) 69–98.
[59] M. van Steen, P. Homburg, A.S. Tanenbaum, Globe: a wide-area distributed system, IEEE Concurrency 7 (1) (1999) 70–78.
[60] M.J. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, The Knowledge Engineering Review 10 (2) (1995) 115–152.
[61] M.J. Wooldridge, N.R. Jennings, Software engineering with agents: pitfalls and pratfalls, IEEE Internet Computing 3 (3) (1999) 20–27.
[62] P.R. Wurman, M.P. Wellman, W.E. Walsh, The Michigan Internet AuctionBot: a configurable auction server for human and software agents, in: Proceedings of the Second International Conference on Autonomous Agents, New York, 1998, pp. 301–308.

Dr. Niek Wijngaards received his M.Sc. degree in Computer Science and his Ph.D. degree in Artificial Intelligence from the Vrije Universiteit, Amsterdam. He is currently an assistant professor at the Vrije Universiteit, Amsterdam. With a strong AI background, his current research focus is to define the types of system support needed to design large scale, reliable heterogeneous multi-agent systems. One example of such support is the provision of agent factories. Design is one domain in which requirements for such support are acquired. All in close collaboration with the Intelligent Interactive Distributed Systems group and the Computer Systems group.

Dr. Benno Overeinder received his M.Sc. degree (cum laude) and his Ph.D. degree in Computer Science from the Universiteit van Amsterdam. He is currently an assistant professor at the Vrije Universiteit, Amsterdam. His research interest is the interdisciplinary area between computer systems and artificial intelligence. This includes scalable middleware layers for multi-agent systems, resource management and security, and applications such as distributed information retrieval. All in close collaboration with the Intelligent Interactive Distributed Systems group and the Computer Systems group.

Prof. Dr. Maarten van Steen is professor of computer science at the Vrije Universiteit, Amsterdam. He has an MS in applied mathematics from Twente University and a Ph.D. in computer science from Leiden University. Van Steen worked at an industrial research laboratory for several years before returning to academia. His research interests include operating systems, computer networks, wide-area distributed systems, and Web-based systems. He is a member of the IEEE and the ACM.

Prof. Dr. Frances Brazier received her M.Sc. degree in Mathematics and her Ph.D. in Psychology, has worked as an associate professor in Artificial Intelligence, and currently heads the Intelligent Interactive Distributed Systems group at the Vrije Universiteit, Amsterdam. She also is research director of the NLnet foundation, that played a major role in raising European wide network infrastructure. Her current research focuses on support for the development of large-scale intelligent, interactive, distributed systems. This includes middleware, services (an agent factory, management services, and directory services), applications (distributed information retrieval and distributed design) but also consideration of legal implications (ALIAS). All in close collaboration with the IIDS group and others. She is a member of the IEEE and the USENIX.