An Awareness Model for Agents in Heterogeneous Environments

Dirk Bade, Lars Braubach, Alexander Pokahr, Winfried Lamersdorf

University of Hamburg, Department of Informatics
Distributed Systems and Information Systems Group
[bade|braubach|pokahr|lamersd]@informatik.uni-hamburg.de

Abstract. One of the constituting characteristics of software agents is their ability to sense the environment. The reception and processing of percepts is a key element for the agent's internal reasoning process and essential for interacting with other entities in the environment. But sensing the environment is often seen as an abstract concept which is practically more or less reduced to the simple processing of some domain-specific message content. In order to be generally applicable among different multi-agent applications a common model of an environment incorporating an extensible set of entities, distribution protocols, and representation- as well as query languages needs to be established. Therefore, we propose a generic, extensible and adaptable model for resource-aware agents. It is organized into different information channels to help directing the focus of interest to specific aspects of the environment. Several discovery- and distribution protocols as well as different representation- and query languages may be used to satisfy the requirements of dynamic environments. The whole model is realized with a dedicated service agent on each platform, which local as well as remote agents can query for environmental information. This way, repeatedly and redundantly integrating these features into every agent application can be avoided and agent developers only have to deal with a simple protocol-API to access the information. Due to our highly flexible and adaptable model, we can face the heterogeneity of multi-agent applications operating in infrastructure- as well as mobile ad-hoc networks.

1 Introduction

Agent definitions often cover the notion of an agent's environment defining an agent as an entity, which is capable of perceiving its environment through sensors and acting upon this environment through effectors [24, 14, 10]. In order to be of any use for an agent or agent application, there has to be a common understanding of what constitutes an environment, how the entities are represented and how the agents can share their knowledge of the environment with each other. Although these aspects may seem natural and trivial for a model, in most of todays agent platforms the environment is only modeled implicitly or even not at all [31].

In order to access some resource, an agent needs to know about its existence, its address and the interaction protocol to use. These details are often hardwired

into the agent's code and hence not adaptable to changes in the environment. In order to deal with dynamic environments, most agent platforms (e.g. FIPA compliant platforms) thus offer yellow-page services (e.g. a directory facilitator, DF [9]), so an agent can lookup appropriate resources at runtime. Drawbacks of this solution are: 1) resources like databases, documents or hardware components cannot be directly registered with existing DFs. They need some kind of service wrapper offered by an agent and 2) in case a required resource cannot be found, an agent has to search for other yellow-page services by itself.

Therefore, this paper proposes a generic, adaptable and flexible model of the environment with extensible knowledge distribution protocols and representations. This model aims to be used in heterogeneous systems, where on the one hand different kinds of entities need to be considered and on the other hand the usage of computational resources (e.g. computing power, network bandwidth, reliability) needs to be adapted to the current execution environment and context. For this reason the presented model may be used in real world agent applications and is appropriate for agents residing on powerful computers in infrastructure networks as well as agents executed on mobile devices in ad-hoc networks.

The rest of the paper is structured as follows: Section 2 introduces the notion of environment and our presented environment model. In sections 3 and 4 different ways of representing and distributing information are addressed. Section 5 introduces our prototypical implementation. The related work part in section 6 highlights other environmental models as well as research efforts in the field of perception, adaptation and ad-hoc networks. Finally, the paper concludes with a subsumption of our proposed solution and our prospect of future work.

2 Environment Model

The environment of an entity is its surrounding, which has an implicit or explicit influence on the entity. It defines the properties and conditions under which an entity exists and provides the processes and principals that govern and support the exchange of information [21]. On the one hand, the environment can be seen as the *execution environment* of an agent, consisting of the execution engine, the agent platform, the message transport system, etc. On the other hand the environment relates to all entities external to the agent platform like the agent's communication partners, a database to work with, a sensor (e.g. measuring the temperature) or documents that are processed by the agent. In contrast to the execution environment, this external view is called the *logical* or *application environment* of an agent [17, 18]. There are also a number of other environmental models that are directed at different aspects of an environment like real world entities or the social links between agents for example [21], but these are not further discussed in this paper.

2.1 Entities and Events

The basis of an environmental model are entities and events. An entity is an abstraction of either an agent, an object or in general a (social) communication

partner [21] and may have several descriptive attributes and a unique identifier to refer to. In [23] a further distinction between goal-oriented (agents) and function-oriented entities (boundary-, resource- and coordination artifacts) is made. While agents are autonomous and social acting entities running on a single node within a distributed system, artifacts offer some kind of function or service, may span over multiple nodes and can be combined to complex artifacts.

Besides the entities we also have to specify, which kind of environmental events may be of interest to an agent. Because a model normally does not represent a static but a dynamic (or in case of e.g. mobile devices and ad-hoc networks highly dynamic) environment, it is not sufficient to know which entities are present, but additionally to be informed once their state changes and whether new entities are available or existing ones disappeared. Such events can be categorized according to their originator into three different classes [18]:

Environment-originating This type of event is caused by the *autonomous process* [21] of the environment, which reflects some kind of external intervention, e.g. the user shutting down an agent platform.

Entity-originating If an entity carries out some action, that leads to an internal state change in some other entity, this state change is said to be entity-originating. E.g. the sending and receiving of a message may set an agent into a busy-state, which possibly needs to be announced to other agents.

Self-originating To this class belongs every event that is not externally caused, e.g. an agent finishing some calculations or the awakening after a timeout.

We are not only concerned about entities but as well about events, because information in a logical environment - in contrast to a natural environment - does not spread automatically throughout the environment. The information that a light switch is turned on for example, is automatically sensed by every real-world entity nearby that is equipped with appropriate sensors (similar to a broadcast). In a logical environment this information has to be explicitly exchanged (unicasted) between the entities by distributing messages containing the new state of the switch artifact. Broadcasts are often not applicable in such an environment, since its scope is restricted to administratively bounded subnets.

In order to model entities as well as entity-related events, we specified a set of requirements that have to be met by an environment model in order to achieve the ability of environmental awareness. These requirements will be introduced in the following.

2.2 Model Requirements

One of the key characteristics of agents is their ability to react to changes in their environment and thereby adapt themselves to the current context. For this reason, agents are often deployed in dynamic environments. In order to provide an appropriate model of the agent's surrounding we derived a set of requirements on the basis of such environments' characteristics. These requirements are as follows:

Awareness Sensing the environment is a continuous process. But as stated above, changes in a logical environment are effectuated by certain events. Our proposed model should therefore be able to detect as well as to proclaim such events.

Heterogeneity The need to deal with heterogeneity relates to two different aspects. On the one hand, we face a multitude of different entities in the environment that somehow have to be represented in the model. On the other hand the model should be used in different infrastructure settings and hence be applicable to servers and mobile devices as well as to infrastructure- and ad-hoc networks.

Adaptivity Supporting the deployment of the model in different infrastructures and dynamic environments also requires to adapt the model to specific conditions. These conditions may affect the way events are processed and proclaimed in the environment. This requirement depends on the ability to be extensible.

Extensibility It should be possible to extend the model in multiple ways. Firstly, the types of entities represented in the model are to be left open, because the model should be usable in a wide range of applications. Secondly, different ways of how to represent an entity should be supported in order to meet the needs of different applications. And thirdly, the way events are proclaimed by the model should be extensible as the infrastructure may require specific forms of information distribution.

Standards The model should adhere to existing standards for representing and distributing knowledge in order to seamlessly integrate information coming from different sources.

Usability Having a model which meets all the above stated requirements, but which can be used neither by a developer nor by any user due to its complexity is inappropriate for open systems. Therefore, the interfaces of the model should be kept simple to allow for easy querying and extending.

In order to meet these requirements, we propose a layered architecture for the environment model, which is presented in the following.

2.3 Architecture

We identified three layers, that focus different aspects of the model: 1) information, 2) representation and 3) distribution. The architecture is depicted in figure 1. First of all, one needs to reason about the types of entities (e.g. remote agent platforms, agents, services or some specific hardware possibly needed for execution), that may be of interest for an agent as well as events, that may occur in an environment, such as (dis)appearing of entities or changes in the state of an entity. Sensors are responsible for gathering information about such events. These aspects are addressed on the upper layer of the architecture. To allow directing the focus of interest to specific kinds of entities, this layer should be organized in a way that combines similar types of entities and events (cf. section 2.4).

The middle layer provides different possibilities to represent an entity. Normally, information about entities is internally stored in a programming language-specific unit (e.g. an object or a structure). In order to distribute this information within a network, these units have to be serializable. But additionally, one might want to query specific fragments of information or use some kind of logic to derive implicit knowledge from explicit information. Therefore, these units have to be transformed into some other representation. For this purpose, different representation and query languages, transformation services as well as the support for ontologies are situated in this layer (cf. section 3).

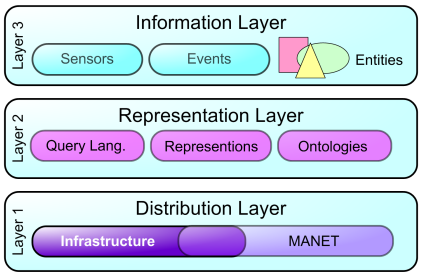

Fig. 1. Architecture of the environment model

The lower layer provides support for different distribution mechanisms. Exchanging knowledge among the agents in the environment is done by using adequate protocols that actively distribute the information. To allow for heterogeneity and the adaptation of the distribution mechanisms to the current context, multiple protocols as well as a generic interface to integrate new protocols are to be provided (cf. section 4). We distinguish protocols for the use in infrastructure networks on the one hand, and protocols especially designed for mobile ad-hoc networks on the other hand.

Before the distribution and representation of information are subsequently addressed, the organization of the upper layer shall be presented in the following. According to the requirements, this layer should manage the entity information in a way that is intuitive to understand for users, easy to extend for developers and simple to query for other agents.

2.4 Information Organization

In order to direct the perception on particular aspects, the information contained in the model is structured in a way, that an agent can choose among different topics. As a metaphor for structuring these kinds of information we used the notion of *channels*. The concept of channels (cf. Microsoft's *Active Channel* [19]) is intuitive for users and suits the needs of our environment model. A channel can be thought of as a FIFO-pipe. Channel news are fed into the pipe at one end, interested entities receive the updates in chronological order at the other end and may further distribute the news on their behalf. An entity claims its

interest by subscribing for a specific channel and hence only receives the desired kind of information, possibly further restricted by using a language-dependent query and a set of constraints.

In order to integrate more topics (e.g. types of entities) into this model, a developer simply has to provide some channel-specific classes and register them with the environment agent. Other agents get to know about these channels when introducing themselves to the environment agent for the first time or by subsequent updates of channels the agent already subscribed for.

2.5 Example

To illustrate the usage of the environmental model and the proposed mechanisms to integrate different distribution and representation methods, an example is presented in the following.

Image Search Imagine an agent, whose task is to search for images given some keywords and constraints. For this task prior knowledge (provided by the user or developer) of one or more image databases is normally required. This may be acceptable for static environments, where the agent has permanent access to a database in its local network or the Internet. But in a dynamic environment the addresses have to be acquired during runtime using some discovery mechanism. But this requires additional effort by the developer, as she has to write the code for accessing different discovery services on her own. Using an environmental model provided by the local agent platform, the agent only has to query the model. It is the task of the model to gather environment information from different sources and present it to the agent on request. If the agent is executed on a mobile device, the model may restrict the ways of gathering information to save transmission costs or even delay the request until an appropriate and low priced Internet access is available.

3 Representing Information

In order to distribute information about the state of the environment and its entities, the agents have to use specific protocols to spread the information. This implies, that the information is represented in a format, which can be serialized and understood by the communicating parties. Most exchange protocols use very simple and proprietary languages to describe entities and their state. Others in contrast apply expressive and standardized languages. But not only the representation is important, but also the ability to query and filter specific information. For example, an agent could only be interested in document artifacts, the existence of other artifacts does not concern the agent and should not be revealed to it. In [7] general insufficiencies and problems of commonly used exchange protocols are described:

Lack of Rich Representations Commonly used languages lack rich representations in order to describe a multitude of different entities and to be able to derive implicit knowledge from the explicit descriptions.

Insufficient Constraint Support In order to reduce the results of a query in some way, the usage of constraints should be supported. Often this feature is not part of a protocol's language.

Vague Matching Most protocols try to match a query against the knowledge base only on a syntactical level, e.g. using string equality. Fuzzy matching or including semantical information would often lead to better results.

Scarce Ontology Support Ontologies could be used to get to a common agreement on terms and statements. When using different exchange protocols in an interaction the usage of ontologies is all the more important as they can help to transform one representation into another.

A simple string-based approach is easy to implement and not very resource demanding, but it suffers from the above mentioned insufficiencies and problems. But since we do not want to restrict the usage of any representation language, our model is generic in a way, that commonly used languages are directly supported and new languages can easily be integrated. To make sure, that two communicating parties understand each other, even if they share no common language, we nevertheless propose a simple, proprietary string-based language as the least common denominator.

In order to support different content languages independent of the used distribution protocol, some kind of content transformation from one language into another has to be done. For example, an agent may request information about some service using an RDF query language (e.g. RDQL [26]), but the results should be returned as a list of simple strings, so that they can be further processed easily. The environment agent therefore has to convert the model into an RDF representation, execute the query and flatten the results to simple strings in order to wrap them in a response message. Such transformation services are an optional part of our model, since this can be a complex task. The usage of ontologies may ease this transformation, but additionally also supports the understanding and interpretation of exchanged knowledge.

4 Distributing Information

A lot of research has been carried out in the field of information distribution and a broad variety of protocols have been specified for this purpose (e.g. Jini [27], UPnP [29], JXTA [28]). These protocols more or less try to find answers to the following questions: 1) how does a newly available entity announces its presence in case it does not know about any other entities? 2) What happens if an entity disappears without explicitly announcing its withdrawal? and 3) How to efficiently distribute information between peers in a way that is scalable, adaptable, reliable and secure? Because our requirements also take the heterogeneity of the infrastructure into account, we raise one more question: How to deal with

all the existing protocols and standards for infrastructure and mobile ad-hoc networks (MANETs) ?

Instead of inventing one more proprietary protocol that suits our needs, the proposed solution adopts existing protocols, that are already in use. Depending on the context the protocol that best fits the current requirements is chosen. In case the agent platform is running on a resource constrained device (e.g. a smartphone) the model additionally has to take care which protocol to choose in order to reduce the amount of used resources (e.g. network throughput, processor cycles, memory) to a minimum.

A protocol serves two different purposes: 1) a protocol may be used to initially find remote entities, e.g. other environment agents to share knowledge with and 2) to distribute knowledge. For example, when an agent is newly created and has the task to find a specific service (e.g. an image database) the developer normally has to tell the agent beforehand how to find such an artifact. In our approach the agent delegates this task to the environment model which in turn may choose among several different protocols in order to find the resource by itself. E.g. it may try to find the artifact directly by sending a multicast, it may use specific protocols to query a registry about an available service or it may contact other agents in order to get some help. Precondition for the latter choice is some kind of address book containing other contacts. When a platform is started, the agent first tries to find a set of initial contacts, i.e. environment agents running on remote platforms, with which it henceforth exchanges information about the environment and any new events. This way, the agent successively also learns about other available contacts and a network of information providers is spanned.

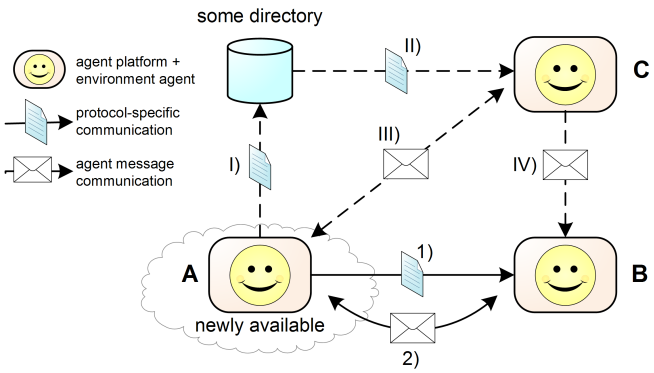

Fig. 2. Contacting remote environment agents

Figure 2 illustrates the architecture and the process of finding contacts in a simplified manner. In a first step a newly available agent platform respectively the newly created environment agent `A` initially tries to find other contacts (interaction `1`). Once a contact has been found the agents may mutually subscribe to information channels and exchange information (interaction `2`). From now on-

wards these two agents inform each other once an event occurs, which impacts one of the channels the agents subscribed for. In parallel, the newly created environment agent may register itself with some kind of directory (e.g. a Jini lookup service, a JXTA peergroup, a directory facilitator, etc.), so that other agents registered with or querying the directory become aware of the new agent (interaction `I`). This way, agent `C` receives the contact details of agent `A` by the directory service (interaction `II`), contacts the agent and exchanges environment information (interaction `III`). As part of the environment information, address details of agent `C` are forwarded to agent `B`, which in turn sends a message to agent `C` to introduce itself. This way, a virtual network is spanned.

4.1 Protocol Requirements

We specified a set of requirements with which we compared different protocols and created some metric in order to infer a proposal for a best-suit protocol in a given context. These requirements were derived from our environment model requirements and aim at the general characteristics of discovery protocols:

Decentralized Operability This requirement focuses on the overall architecture of a protocol. In general we distinguish between client/server- and peer-to-peer protocols. Especially for unreliable MANETs a peer-to-peer approach is vitally important, because of the absence of a single-point-of-failure. But also scalability and the locality of information have to be taken into account.

Interoperability This point comprises two different aspects. Due to heterogeneity, protocols and their implementation respectively should be inter-platform as well as inter-protocol operable. Platform-interoperability is achieved either by being available for different operating systems or by being interpreted by a virtual machine. Protocol-interoperability means that a protocol either uses a standardized representation format or that proxies or bridges are available to bridge the syntactic gap between two or more protocols.

Awareness Support The term *awareness* needs to be distinguished from the terms *lookup* and *discovery*. While the latter ones describe a single action, awareness refers to a continuous process, where state changes are pushed to interested entities rather than actively pulled periodically [16]. When pushing state changes the receiver is instantly informed about the new state and any inconsistencies between the actual state of the environment and the state of the model may be corrected early.

Lease Mechanisms Such mechanisms are used in order to prevent a system from finally being blocked by outdated information. Without employing lease mechanisms withdrawing entities would have to explicitly deregister upon leaving the system. Since a withdrawal might not necessarily be intended (e.g. unexpected connection aborts) one cannot rely on proper deregistrations and hence some kind of lease should be supported. This feature is especially important for highly dynamic environments (e.g. MANETs).

Resource Demands Considering a highly dynamic environment as well as resource constrained devices, protocols should only require a minimum of mes-

sages being exchanged and computational resources being used. This requirement also comprises the need for being scalable.

Scope Some protocols rely on multicast-messaging, specific routing-protocols or e.g. DHCP-server for configuration settings, and are therefore restricted to being used within local subnets. Other protocols are technology-dependent in a way, that also restricts their application to locally or administratively bounded networks (e.g. Bluetooth SDP). In order to be used in an Internet-scale distributed system, a protocol should therefore support standard Internet-protocols. Optionally limiting the scope may be desired in order to restrict traffic to a reasonable amount.

Representation/Filtering Information about entities must be represented in a serializable format in order to be distributed in a network. Several standards exist for this purpose, starting from flattened string representations to expressive logical descriptions. But information must not only be represented, but a protocol must also be able to filter information beforehand in order not to cause too much unnecessary traffic and resource exhaustion.

With these requirements we evaluated some of the most promising protocols for infrastructure as well as mobile ad-hoc networks [1]. Table 1 gives an overview of our results (restricted to infrastructure protocols). A similar evaluation of protocols to be used in MANETs (e.g. *Bluetooth SDP* [4], *Konark* [11], *DEAPspace* [20], *Card* [12], *Scalable Service Discovery for MANET* [25], etc.) has also been done. But most protocols are specifically designed to be used in MANETs with certain characteristics and it is therefore difficult to compare these protocols with each other. The results of our evaluation show that all of the protocols have their advantages and disadvantages in certain areas, which corroborates our approach of adaptively choosing an appropriate protocol at runtime depending on the context.

	Decentralized Operability	*Inter-operability*	*Awareness Support*	*Lease-Mechanism*	*Resource-demands*	*Scope*	*Represent./ Filtering*
Jini	O	O	+	++	-	O	O
SLP	O	+	- -	+	O	-	+
UPnP	+	+	+	- -	-	-	+
Salutation	++	O	O	- -	O	O	+
JXTA	++	++	- -	+	O	++	+
	- - very bad, - bad, O sufficient, + good, ++ very good						

Table 1. Evaluation of distribution protocols for infrastructure networks[1]

5 Prototypical Implementation

In order to prove that our approach is realizable, we implemented a prototypical component for the *Jadex BDI Agent System* [5]. The component is responsible for creating and updating the model of the environment as well as to make the

information available to any local or remote agents. The component itself (called *resource facilitator*, RF) is realized on the application level as a service agent, running on the agent platform[1] (comparable to the FIPA directory facilitator for example). This way, only one instance of the environment model needs to be managed on every platform.

In a first step we chose the objects, contained in the model, to be the set of possibly interesting entities, that an agent might want to be aware of. These are specific for every device and are categorized as follows:

Hardware The hardware resources of the device, like e.g. processor, memory, network interfaces, screen, storage and their corresponding attributes like capacity or current workload.

Software Besides the hardware, the software infrastructure of a device may also be of interest. Therefore, the model contains a set of properties of the agent platform, the virtual machine and the operating system.

Location If possible, a device also offers some location information. This may be in the form of a descriptive statement (for the user), a network address, GPS coordinates, etc. and may be used for example, to choose a nearby resource among a multitude of offered resources.

Services If an agent offers a service, this is normally registered with a local or remote directory facilitator (DF). Other agents looking for this service have to know the address of the DF in order to get the contact details of the provider. Information about services as part of the environmental model is automatically distributed within the environment and is available for every interested party without the need to know the directory where the service has initially been registered.

Agents As agents often want to communicate with other agents in order to cooperate, they would benefit of information about possible communication partners. This way an agent could dynamically find counterparts without the need for the user providing contact details.

Heartbeat A heartbeat can be thought of as an abstract entity and is therefore included in the environment model as an alternative to lease mechanisms (further described below).

As stated earlier we use a channel abstraction to query and distribute the information. The left-hand side of figure 3 depicts the organization of the model. Information coming from several local information providers is fed into the appropriate channels. Each channel internally processes the information independently for every subscriber. Processing incorporates the execution of queries, the application of constraints (e.g. limiting the result set) and the transformation into a desired representation language.

Noteworthy is the above mentioned heartbeat channel offered by every RF. Using this channel one can force a remote RF to periodically sent a heartbeat (as long as no other message is sent). This way, one can make sure that the whole

[1] Currently only the Jadex standalone platform as well as Jadex' adapters for JADE [2] and DIET [15] are supported.

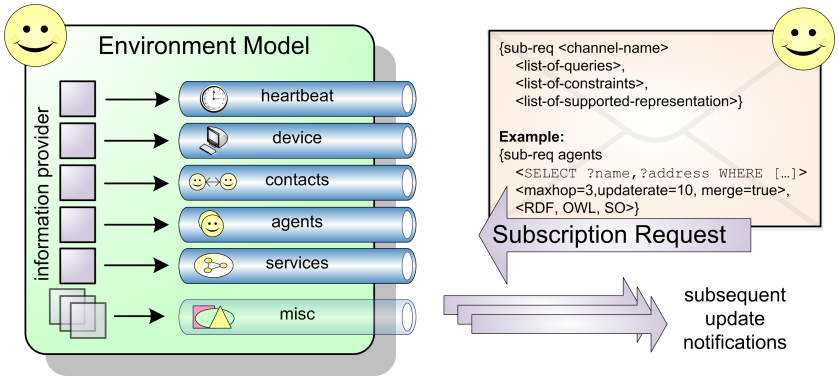

Fig. 3. Organization of the model into channels

agent platform is still running. Combined with other channel subscriptions the integrity of nearly every entity residing on a remote platform can be monitored. For example, to make sure a remotely offered service is available, one can subscribe to the heartbeat channel and the service channel. When no heartbeat is received, the platform is supposed to be unreachable and hence the service. In case the platform is still running, but not the service, the RF would have posted the information through the services channel. As a consequence the heartbeat circumvents the need for lease-times for entity information. Information about unreachable devices and their resources respectively is subject for the model's internal garbage collection.

In the given example (figure 3) an agent subscribed for the *agents*-channel in order to get to know about any other agents in the environment. The subscription request contains an *RDQL*-query focusing only on the name and address of other agents as well as a set of constraints to further straighten the results and finally an ordered list of supported representation languages in one of which the result shall be transformed.

When a platform is started, the RF initially collects information about the local resources by evaluating attributes provided by the user or the virtual machine and by asking the local directory facilitator and the local agent management system. In order to gather information about the external environment and its entities the RF has to query remote RFs and subscribe to specific channels. For this purpose, we designed a simple subscription protocol and multiple methods for initially finding remote RFs (e.g. by multicast, by other contacts and by protocol specific first-contact mechanisms, e.g. by joining dedicated JXTA peergroups or by requesting a Jini lookup service). Additionally, each RF maintains a list of formerly known contacts which also might be contacted. The subscription protocol is a two-way protocol, allowing RFs to introduce each other. During the initiation phase the RFs exchange device information as well as information about offered channels, distribution protocols and supported content languages.

Additionally, we designed a generic interface for distribution protocols. In order to exchange and query channel information an adapter needs to be written for every supported protocol. Such an adapter takes the information to be

distributed as well as several attributes like update events, update intervals and addressees as input and sends the information on demand or at specific points in time to each subscriber.

In order to deal with the different content languages additional subcomponents are needed. Such a component takes the information stored in the environment model and transforms it into an appropriate representation. This way, Java objects are mapped into an RDF or OWL syntax, for example, which is afterwards handled by a protocol specific message wrapper. Additionally, the RF supports query languages in order to further reduce the information that is sent via a specific channel. For example, one might only be interested in agents, whose names have a predefined prefix or devices, which have powerful processors or a broadband Internet connection.

In a last step, we parameterized all the mechanisms in order to gain adaptation during runtime. For example, when the throughput of the network connection decreases, model updates could be made more infrequently and low-bandwidth protocols could be used instead of the highly demanding ones. Further on, language specific information merger are supported, which may be used to merge delayed channel updates (e.g. no update is sent when a newly created agent is immediately terminated). In addition we implemented a message pool, storing channel updates for each addressee so that these can be merged before they are finally sent as a single message.

The RF-component as well as several subcomponents have already been implemented. The implementation is compatible with the *J2SE v1.3* and the *J2ME Personal Profile v1.0* and may therefore also be executed on mobile devices.

6 Related Work

A lot of research has already been carried out in the areas affecting this work. Agents and their environment for example have been investigated by numerous researchers. Russel and Norvig [24] as well as Ferber [8] discuss a number of key properties for classifying environments. Furthermore, Odell [21] distinguishes between the physical- and the communication environment, lists required principles and processes and also considers spatial and temporal aspects. A survey of the state-of-the-art environment models can be found in [30], where additionally the aspect of mobility and associated *place* and *region* abstractions are taken into account. Unfortunately, none of these authors aim at a practical design of their proposals and concentrate on the conceptual level. Another prominent model are the artifact and coordination context abstractions by Ricci and Omicini [22]. Their approach has some similarities with ours, but concentrates on the exploitation of artifacts supporting the coordination of agents, while we focus on the infrastructural support for exchanging artifact representations in general.

Mertens et al. [17] address the adaptation in a multi agent system for example, but in contrast to our approach the environment adapts itself to the applications' needs, which is practically impossible in the open distributed systems we address. A need for adaptation in dynamic environments is also backed

by Chen and Kotz [6], who state that context-aware applications will be more effective and adaptive to users' needs. For the application of agents in mobile ad-hoc networks standardization proposals have been published by FIPA [3] and Lawrence [13] in 2002. Unfortunately, work on this topic seems to have been discontinued by FIPA.

The reason for designing a new model instead of adapting one of the existing approaches is twofold. Firstly, most of the research has been done at a conceptual level and is not suitable for a concrete design or implementation. Secondly, the few existing environment models that agent platforms offer, are targeted either at specific application domains or infrastructures and are therefore not applicable in heterogeneous environments. Additionally none of the models found in literature addresses the organization of the model and the distribution of the knowledge among the agents as well as representation and query languages for information.

7 Conclusion and Future Work

This paper introduced an environment model for multi-agent systems, which provides agents with an impression of accessible entities in their logical surrounding. The entities may be other agents or some kind of functional artifact. We opposed no restrictions on the types of entities, whose representations are contained in the model. Although we suggested a common basis of entities (hardware, software, agents and services) to be included, the model is easily extensible to integrate other types of entities (e.g. documents, real-world entities, etc.) as well. In order to focus specific aspects when sensing the environment, the model is organized in different information channels, which an agent can subscribe to. The information is then distributed actively between the agents in the environment by adaptively choosing one or more distribution protocols. The choice is up to the environment agent, responsible for managing the model, and depends on the context, which is made up by the user's preferences and the supporting hardware infrastructure. Freedom of choice also holds for the kind of representation used for describing entities. A generic interface allows for using description languages ranging from simple string-based languages to complex logics.

Due to the flexibility of the proposed model it is suited to be used in heterogeneous environments, where on the one hand different types of entities need to be represented and on the other hand various execution platforms, ranging from resource constrained mobile devices to fully equipped servers need to be considered. Because the model relies on different distribution protocols and representation languages to choose from, it can adapt itself to changing environmental conditions and thus deal with the dynamics of an environment at runtime.

Our prospect of future work is directed at multiple improvements. One aspect is the integration of privacy policies making it possible to restrict the access to certain information. Another aspect aims at different roles for environmental agents. Most existing peer-to-peer networks are backed by some kind of *supernodes*, which process more traffic than others in order to unburden resource constraint nodes in the periphery. We are adopting this approach to enhance

scalability by introducing a contact-based overlay network in which some agents have a more global view on the environment than others.

Our current research efforts point at the question, of what to do with an environment model. One very promising research field are mobile agents, since these depend on sensing the environment and choosing appropriate execution platforms. An environment model could provide better arguments for migrating on some device than a user may possibly do, especially in dynamic environments. In the prospect of mobile computing, ubiquitous computing and ambient intelligence, this could be a very promising field of research.

References

1. D. Bade. Kontextabhaengige und eigenverantwortliche migration von softwareagenten in heterogenen umgebungen. Master's thesis, Uni Hamburg, 2007.
2. F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.
3. M. Berger and M. Watzke. Agents in ad hoc environments. Technical report, FIPA, December 2002. FIPA00068.
4. Bluetooth. Bluetooth specification v.1.1. www.bluetooth.com, February 2001.
5. L. Braubach, A. Pokahr, and W. Lamersdorf. *Jadex: A BDI-Agent System Combining Middleware and Reasoning*, pages 143–167. Whitestein Series in Software Agent Technologies, Birkhuser Verlag, 2005.
6. G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Hanover, NH, USA, 2000.
7. H. Chen, A. Joshi, and T. Finin. Intelligent agents meet jini in the aether. *Cluster Computing*, 4(4):343–354, October 2001.
8. J. Ferber. *Multiagentensysteme - Eine Einfhrung in die Verteilte Knstliche Intelligenz*. Addison-Wesley, 2001.
9. FIPA. Fipa abstract architecture specification, December 2002.
10. B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, 1995.
11. S. Helal, N. Desai, V. Verma, and C. Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. *WCNC 2003, IEEE*, 3:2107–2113, 03. 2003.
12. A. Helmy, S. Garg, N. Nahata, and P. Pamu. Card:a contact-based architecture for resource discovery in wireless ad hoc networks. *Mob. Netw. Appl.*, 10(1-2), 05.
13. J. Lawrence. Leap into ad-hoc networks. In *In Proc. of the Ubiquitous Computing Workshop, Bologna, Italy*. Media Lab Europe, 2002.
14. P. Maes. Artificial life meets entertainment. *Commun. ACM*, 38(11):108–114, 95.
15. P. Marrow, M. Koubarakis, and R. van Lengen. Agents in decentralised information ecosystems: The diet approach. In *Proccedings of the Symposium on Information Agents for E-Commerce, AISB Convention*, York, UK, March 2001.
16. R. E. McGrath. Discovery and its discontents: Discovery protocols for ubiquitous computing. Technical report, University of Illinois, Department of Computer Science, Champaign, IL, USA, 2000.
17. K. Mertens, T. Holvoet, and Y. Berbers. Adaptation in a distributed environment. In *Environments for Multiagent Systems*, pages 49–59, 2004.
18. K. Mertens, T. Holvoet, and Y. Berbers. A case for adaptation of the distributed environment layout in multiagent applications. In *SELMAS '05: Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems*, pages 1–8, New York, NY, USA, 2005. ACM Press.

19. Microsoft. Introduction to active channel technology. http://msdn.microsoft.com.
20. M. Nidd. Service discovery in deapspace. *IEEE Pers. Comm.*, 8(4):39–45, 2001.
21. J. Odell, H. V. D. Parunak, M. Fleischer, and S. Brueckner. Modeling agents and their environment. In F. Giunchiglia, J. Odell, and G. Weiß, editors, *AOSE*, volume 2585 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2002.
22. A. Omicini, A. Ricci, and M. Viroli. Coordination artifacts as first-class abstractions for mas engineering: State of the research. In *Software Engineering for Multi-Agent Systems IV*, pages 71–90, 2005.
23. A. Ricci, M. Viroli, and A. Omicini. Programming mas with artifacts. In *Programming Multi-Agent Systems,3rd Int. Workshop,ProMAS*, pages 206–221, 2005.
24. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.
25. F. Sailhan and V. Issarny. Scalable service discovery for manet. In *PERCOM '05: Proc. of the 3.IEEE Int. Conf. on Perv. Comp. and Comm.*, pages 235–244, Washington, DC, USA, 2005. IEEE Computer Society.
26. A. Seaborne. Rdql - a query language for rdf. http://www.w3.org/Submission/RDQL/, Jan. 2004.
27. Sun. Jini architecture spec. v. 2.1. Technical report, Sun Micro., Inc., Dec. 2001.
28. Sun. Jxta v2.0 protocols specification. https://jxta-spec.dev.java.net/, 2007.
29. UPnP. Universal plug and play device architecture version 1.0.1. http://www.upnp.org, December 2003.
30. D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems. In *E4MAS*, 04.
31. D. Weyns, G. Vizzari, and T. Holvoet. Environments for situated multi-agent systems: Beyond infrastructure. In *E4MAS*, volume 3830 of *Lecture Notes in Comp. Sci.*, pages 101–117. Springer, 2005.