The Notions of Application, Spaces and Agents

New Concepts for Constructing Agent Applications

Alexander Pokahr, Lars Braubach

Distributed Systems and Information Systems,
Computer Science Department, University of Hamburg

Introduction

Agent-based approaches offer promising solutions for recent demands in enterprise software like adaptability cooperation and coordination (Jennings 2001). Industrial strength agent platforms, such as the JADE platform or Whitesteins LS/TS tool suite, serve as an established technological basis for agent-based programming and software development (Unland et al. 2005).

While these systems consider agents as first-class entities, they currently offer only limited conceptual and technological support for 1) dealing with the agent application as a whole and 2) coping with other non-agent components of an application (i.e. the agent environment). As a result, today's agent-based software systems often employ ad-hoc solutions for these issues, suffering from a lack of tool support as well as poor maintainability.

The concepts and tools presented in this paper allow for defining an agent application as a conceptual entity, which is composed of agents as well as additional non-agent components. A pluggable architecture is proposed that allows defining common types of environment models, thereby facilitating seamless integration of the corresponding environment components into an agent application. The architecture is implemented as part of the Jadex agent framework.[1]

The paper is organized as follows. In Section 2 the conceptual framework for application description as well as the pluggable environment architecture are presented. Furthermore, as an example, the environment support space is presented. In Section 3 an example application is presented that illustrates the advantages of the proposed framework. Section 4 discusses related work and Section 5 concludes the paper with a summary and an outlook.

[1] http://jadex.informatik.uni-hamburg.de

Application Description and Pluggable Environments

As defined in (Wooldridge 2001, p. 3) "a multi-agent system is one that consists of a number of agents, which interact with one another [...]." This represents the classical agent-centered view of what an agent application is. It consists of interacting agents and provides its functionalities as a result of their coordinated action. This definition highlights that historically the agent has been considered as the only core concept of multi-agent systems. Recently, this view has shifted towards richer conceptual models, which add further first class entities to the agent paradigm. Most of these approaches consider closer what makes up the agent's environment and try to introduce respective concepts. E.g. Weyns et al. (2007) propose that the environment itself should be a first class entity, whereas in the A&A paradigm (Ricci et al. 2007) finer grained concepts like workspaces and artifacts are introduced.

One question that arises from these approaches is how agent platforms can generically support the construction of these extended and enhanced agent paradigms without strongly committing to one specific approach. Hence, in this section an attempt is made to consider the agent application itself as first class entity and allow an easy and especially extensible way to specify what this application is composed of. Besides the extensibility another important requirement is that an application description should abstract away from details of the agent level, i.e. it should not make any assumptions about the agent types used and their internal structure. In the following, details of the application specification concepts are explained.

Concepts for Application Specification

In general, an application specification mainly contains information about the application type, i.e. about the underlying structure of runtime elements similar to e.g. a class definition in object-oriented languages.

Naturally, one core concept of the application type definition is that of an agent type. In addition, so called space types are introduced, which have been inspired by the context and projection concepts of the Repast simulation toolkit (cf. Section 4). A space is a very general concept for the representation of non-agent elements. It is a structure that contains application specific data and components, which are independent of a single agent. Therefore a space provides a convenient way of sharing resources among agents without using purely message-based communication. The space concepts can be seen as an additional means for structuring. It does not impose constraints on the agents, i.e. agents from different applications can communicate via messages as usual.

Spaces also can be seen as an extension point of the agent platform as spaces offer application functionality, independent of agent behavior. Please note that the concrete functionalities of a space depend on its concrete type and are not directly part of the application concepts.

In order to define in what way an application instance should be created from an underlying application type, the concept of configurations is introduced. A configuration describes which runtime entities comprise a specific application instance, i.e. which agents and spaces should be created at startup time.

At runtime an application represents an entity in its own right, which mainly acts as a container for agents and spaces. Agents that are part of an application can access the spaces via their application instance. In this way the access to spaces is restricted to agents from the same application context. Representing applications as entities also allows for handling them at the tool level, i.e. instead of starting or stopping single agents, whole applications can be managed.

Application Descriptor

In Jadex, applications are described using an XML descriptor file. The syntax of an application descriptor has been defined using an XML schema. In order to explain the details of the descriptor a cutout of this schema is depicted in Figure 1.

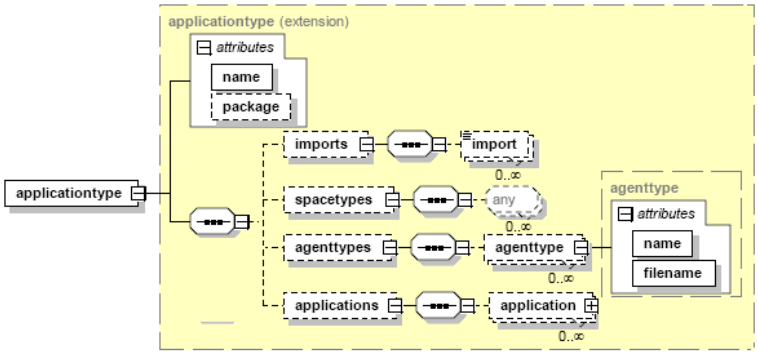

Figure 1: XML-schema for application descriptors

The cutout shows that an application type is comprised of definition sections for space types, agent types and applications. Additionally, an imports section allows for including files and complete packages from other directories (similar to the Java import mechanism). An application type is defined by a name, package and a declaration of necessary namespaces, e.g. for certain space types. The space type section may contain an arbitrary number of space type definitions. As space types are meant as an extension point, the XML schema "any"-element is used. This allows for concrete space types being defined in their own XML schema and to be included at this point using their own tag structure.

The agent types of an application are defined in the respective section using only a symbolic name, which is used to identify the type and a filename, which points to the file in which the corresponding agent type is defined. In this way the application descriptor remains agnostic with respect to the internal agent architecture that was used for developing the agent (e.g. BDI or a task-model architecture).

Finally, in the application section different start configurations can be defined as application (cf. Figure 2). An application can be given a name and may contain any number of space and agent instances. According to space types, also space instance specifications are left open for external schema definitions and can hence be included in a customized fashion. Nonetheless, all space implementations must adhere to a specific implementation interface, which allows the application to automatically instantiate the space at runtime. An agent is specified using several parameters and arguments from which only the type is mandatory. It represents a reference to one of the defined agent type names from the application model. Furthermore, an instance name, a start configuration, a number, as well as start and master flags can be set. The instance name is just the name for the agent being created, while the start configuration allows starting an agent with a specific predefined setting. The number describes how many agents of the same type will be initialized and is thus an efficient shortcut notation. Finally, the start flag allows deciding if agents should only be created or created and also started, whereas the master flag can be used for tagging essential application agents. The deletion of such a tagged agent will lead to the termination of the whole application.

In addition, agent instances can also be supplied with an arbitrary number of named arguments, which are defined as Java expressions. At startup the arguments will be provided to the agent and are processed according to the agent architecture (e.g. BDI agent will use arguments as initial values for corresponding beliefs).

Virtual Environment Support

In the preceding sections it has already been mentioned that one core extension concept for agent applications proposed in this paper are spaces. In order to illustrate this concept, in this section the environment support space will be explained in more details. This space is meant to be a virtual 2d environment for situated agents, in which they can perceive and act via an avatar object connected to them. The space facilitates the construction of simulation examples, as it takes over most parts of visualization and environment agent interaction.

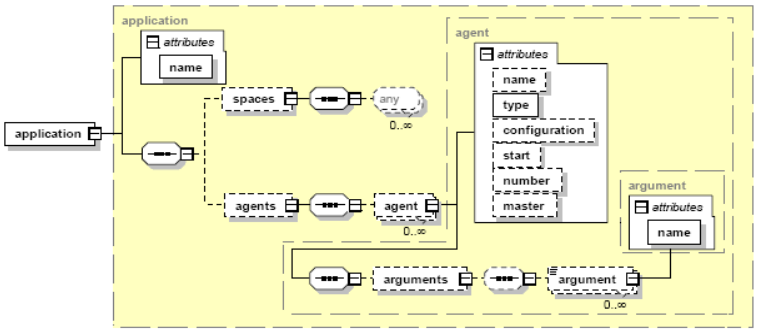

Figure 2: XML-schema application instance specification

Please note that the environment space is only one very specific kind of space that has been developed. The space concept in general can be interpreted in completely other ways, e.g. in an organizational or deontic way. Two further spaces, that already have been developed, are a simple version of Ferber's agent-group-role model (cf. Ferber et al. 2003) and a preliminary version of a connection space for environment interface standards approach (Behrens et al. 2009). Another space type that is currently under development is a coordination space for weaving decentralized coordination mechanisms in the application without changing the agent's behavior descriptions.

The environment support is quite elaborated and can thus not be described thoroughly in this paper. Basically, it assumes an environment to be a 2d area (either a grid or a continuous space), in which space objects are located at certain positions. Specific space objects, called avatars, are connected to agents and allow them to act and perceive in the environment via user defined actions and percepts. Furthermore, tasks can be directly attached to space objects. They represent the behavior of an object, which is automatically executed by the space as time advances. An example for a typical task is a movement to some target coordinates. The task will the continuously compute and adapt the object's position according to the speed and time progress. Besides object behavior, also global behavior can be specified in terms of environment processes, which may operate on all objects of the environment. Such processes can e.g. be used to model environmental activities, like heat diffusion or gravitation forces.

Using the aforementioned concepts the application domain can be described. In addition, also the visualization can be specified in terms of possibly different perspectives. A perspective basically consists of drawables, which specify a graphical representation of a space object type. Besides the basic form, which can be e.g. a geometrical shape or an image, also many further refinements are possible. One can e.g. compose a drawable from other drawables and use draw expressions for deciding about the size, rotation and appearance of the object. Hence, it is e.g. possible to change the view of an object if it carries an item.

In the following section an example application will be presented that further explains how application descriptors can be used and additionally how the flexible space concept can be exploited for simulating and executing the target scenario.

Example Application: Hospital Scheduling

The application development concepts and tools described in this paper are illustrated in the following by using an example application from the medical domain. In the MedPAge-Project (Medical Path Agents), as a part of the German priority research program "Intelligent Agents and Realistic Commercial Application Scenarios" (SPP 1083), the usefulness of agent technology for building hospital scheduling systems has been explored (Paulussen et al. 2006; Zöller et al. 2006). In the

course of this project, several prototype applications have been realized to benchmark agent-based scheduling algorithms against the status-quo as well as evaluating the acceptability of such a system with the aid of hospital personnel.

The goal of the MedPAge system is to provide solutions to the scheduling problem of continuously assigning hospital resources (e.g. a radiology unit) to patients requiring treatments. The system follows an agent-based approach: All resources and patients are represented by corresponding agents, which negotiate to achieve a schedule that meets hospital goals (high utilization of resources) as well as patient goals (short waiting times). The adaptability of the approach has proven to be very useful in the highly dynamic and uncertain domain of hospital operation, where e.g. expected treatment durations are often exceeded due to complications and schedules become invalid due to the arrival of emergency patients.

Application Definition

An explicit application definition allows separating the agent implementation from the details of the application environment. This means that for the MedPAge system a clean conceptual separation between the scheduling algorithm, implemented in patient and resource agents, and the hospital environment can be achieved.

In the application descriptor 'medpage.application.xml' (see Figure 3) you can see that a separate namespace for the environment elements is declared (line 2). The hospital space type (lines 6-19) defines the environment that the scheduling agents will act in. For easy understanding, only the declarations relevant for the agent/environment interface are included (percepts and actions). The details of a concrete environment implementation are discussed in section 3.2. There is only one action in the system, 'propose next patient' (line 9), which is performed by resource agents, after they finished the negotiation for the next time slot. Note that the system could operate autonomously, but for acceptability reasons the system only makes recommendations, which are manually confirmed by hospital personnel. The required information for the negotiations is fed into the agents by the respective percepts (lines 12-17). Both patient and resource agents are notified, whenever the patient or resource they are representing becomes free or occupied (lines 12-15). For a resource, the 'occupied' state represents resources performing scheduled treatments as well as resources, which are blocked due to emergency treatments or repair. An 'occupied' patient is probably having a treatment at the moment, but could also be unavailable for other reasons, e.g. due to visitation.

```
01: <applicationtype xmlns="http://jadex.sourceforge.net/jadex-application"
02:   xmlns:env="http://jadex.sourceforge.net/jadex-envspace"
03:   name="MedPAge" package="medpage3">
04:   <imports>...</imports>
05:   <spacetypes>
06:     <env:envspacetype name="hospitalspacetype" …>
07:       ...
08:       <env:actiontypes>
09:         <env:actiontype name="propose_next_patient" agenttype="ResourceAgent" …/>
10:       </env:actiontypes>
11:       <env:percepttypes>
12:         <env:percepttype name="resource_free" agenttype="ResourceAgent"/>
13:         <env:percepttype name="resource_occupied" agenttype="ResourceAgent"/>
14:         <env:percepttype name="patient_free" agenttype="PatientAgent"/>
15:         <env:percepttype name="patient_occupied" agenttype="PatientAgent"/>
16:         <env:percepttype name="treatment_required" agenttype="PatientAgent"/>
17:         <env:percepttype name="treatment_done" agenttype="PatientAgent"/>
18:       </env:percepttypes>
19:     </env:envspacetype>
20:   </spacetypes>
21:   <agenttypes>
22:     <agenttype name="PatientAgent" filename="medpage3/agent/patient/PatientFCFS.agent.xml"/>
23:     <agenttype name="ResourceAgent" filename="medpage3/agent/resource/ResourceFCFS.agent.xml"/>
24:   </agenttypes>
25:   …
26: </applicationtype>
```

Figure 3: MedPAge application descriptor (application definition cutout)

The actions and percepts are linked to the logical agent types 'ResourceAgent' and 'PatientAgent'. The concrete implementation of these agents is specified in the agent types section (lines 22, 23). The agents are implemented as Jadex BDI agents, which are declared in separate xml files. Using different agent implementations in this place, the scheduling algorithm can easily be replaced. In the figure, the FCFS (first-come-first-serve) algorithm is used.

Environment Implementation

The application descriptor not only declares the agent/environment interface, but also their implementation. The agent implementation refers to external files that are specific to the agent model (e.g. BDI, cf. Section 3.1). The environment implementation is included in the environment space type definition. Figure 4 shows an environment that allows testing MedPAge scheduling algorithms in a simulation setting. Two types of virtual entities are defined; 'patient' (lines 3-7) and 'resource' (8-12), which each are characterized by their properties. The most important property of a virtual patient is the clinical pathway, i.e. the sequence of treatments, which are required until the patient can be dismissed. The resource is characterized by a type, which determines the types of treatments that the resource can perform.

The behavior of the virtual environment is implemented in tasks performed for each object and global processes. The patient iteration task (line 15) is started for each patient and uses the clinical pathway of the patient object to issue the 'treatment required' percepts at appropriate times. Two processes are furthermore responsible for the overall simulation progress. The 'patient arrival' process (line 18) creates new patients based on an exponential distribution. Moreover, each newly created patient is assigned a generated clinical pathway, based on real statistical data containing 3,448 data sets with information on medical tasks for 792 inpatients from admission to release (Zöller et al. 2006). The 'resource operation' process performs the scheduled treatments as proposed by the resource agents. Treatment durations are simulated based on random distributions from the data set. The process also issues the 'occupied' and 'free' percepts for the corresponding patient and resource, whenever a treatment starts or completes. The 'propose next patient' action already described in the last section is now mapped to a concrete implementation (line 22). The action is implemented in a Java class, which in this case forwards the proposal to the resource operation process.

Different hospital configurations can be specified at the instance level in the applications section. Resource instances of type 'RAD' for a radiology unit and of type 'ENDO' for an endoscopy unit are declared in the 'default' application (lines 27-47). As multiple application configurations can be declared in the descriptor, different hospital scenarios can be specified and chosen for testing a scheduling algorithm. For each resource object, a resource agent is automatically created.

Summary

Development requirements that occurred during the course of the MedPAge project can nicely be tackled with the proposed concepts. Separating the algorithm implementation in the agents from the environment configuration in the application description facilitates testing of the scheduling algorithms in different scenarios by varying, e.g., the hospital size or patient arrival rates. Also benchmarking alternative algorithms is easy by exchanging the agent implementations (lines 22

and 23 in Fig. 3). The clean separation between agents and the environment allows switching from a simulated environment to an operational prototype. In the Med-PAge project, such a prototype was used in an acceptability evaluation study with practitioners and included the agent-based scheduling implementation, but not the simulated environment. Instead, patient data was entered by hospital personnel.

```
01: <env:envspacetype name="hospitalspacetype">
02:   <env:objecttypes>
03:     <env:objecttype name="patient">
04:       <env:property name="name "/>
05:       <env:property name="clinical_pathway"/>
06:       …
07:     </env:objecttype>
08:     <env:objecttype name="resource">
09:       <env:property name="name"/>
10:       <env:property name="type"/>
11:       …
12:     </env:objecttype>
13:   </env:objecttypes>
14:   <env:tasktypes>
15:     <env:tasktype name="patient_iteration" class="PatientIterationTask"/>
16:   </env:tasktypes>
17:   <env:processtypes>
18:     <env:processtype name="patient_arrival" class="PatientArrivalProcess"/>
19:     <env:processtype name="resource_operation" class="ResourceOperationProcess"/>
20:   </env:processtypes>
21:   <env:actiontypes>
22:     <env:actiontype name="propose_next_patient" agenttype="ResourceAgent" class="ProposeNextPatientAction"/>
23:   </env:actiontypes>
24:   …
25: </env:envspacetype>
26: …
27: <application name="default">
28:   <spaces>
29:     <env:envspace name="hospitalspace" type="hospitalspacetype">
30:       <env:objects>
```

```
31:       <env:object type="resource">
32:         <env:property name="name">"Radiology 1"</env:property>
33:         <env:property name="type">"RAD"</env:property>
34:       </env:object>
35:       <env:object type="resource">
36:         <env:property name="name">"Radiology 2"</env:property>
37:         <env:property name="type">"RAD"</env:property>
38:       </env:object>
39:       <env:object type="resource">
40:         <env:property name="name">"Endoscopy 1"</env:property>
41:         <env:property name="type">"ENDO"</env:property>
42:       </env:object>
43:       ...
44:     </env:objects>
45:   </env:envspace>
46:  </spaces>
47: </application>
```

Figure 4: MedPAge application descriptor (environment definition cutout)

Related Work

Our work deals with the explicit definition of agent applications and agent environments. The definition of agent applications has to some extent already been considered in the area of multi-agent platforms. E.g. platforms like Jason[2] and 2APL[3] offer MAS (multi-agent system) description files for starting a multi-agent system. In Jason the MAS description allows for specifying the agent instances that should be started and the environment instance that should be initialized at start-up. Agents and environment are directly specified via their file names. Similar to the approach presented in this paper it is assumed that an agent application is comprised of agents and an environment, but the environment is seen as a black box that cannot be configured in the application description.

Agent application configuration and deployment has been researched in the ASCML (agent society configuration manager and launcher) project (Braubach et al. 2005). In this project a reference model and tool support was presented that allows distributed deployment of agent applications. In an ASCML application descriptor, the agents belong to an application and their dependencies can be spe-

[2] http://jason.sourceforge.net/

[3] http://www.cs.uu.nl/2apl/

cified. The ASCML is targeted towards purely agent-based applications and therefore does not consider non-agent components. The basic structure of our application descriptor is similar to the ASCML, including agent types and instances as well as application configurations. Our model adds the notion of space for including non-agent components. Unlike the ASCML model, we currently do not support the explicit specification of agent dependencies in the application description. The model presented here is simpler but more flexible and extensible.

Several other approaches introduce the environment as a first-class entity in application development. Most notably, simulation toolkits like SeSAm[4], Repast[5] or NetLogo[6] provide an infrastructure for creating a virtual environment that the simulated agents can be executed in. These toolkits are aimed at developing simulation models and often assume that one specific kind of environment exists, which can be directly manipulated by the agents. In this respect the coupling between the virtual environments and the agents is very tight and the description of simulation models does not aim at incorporating different or multiple environments. Among the simulation toolkits Repast is an exception, as it provide with contexts and projections two very powerful and flexible concepts. A context is meant to be a container for agents, while a projection is some kind of structuring on a context. Our model has been inspired by the Repast concepts and simplifies them in the following way. We assume that the only context for agents is the application itself. Furthermore, spaces are similar to projections with the difference that a projection is considered being a passive relationship of agent within a context. The meaning of spaces is broader as their content is not restricted. For example a space may also encapsulate a whole virtual world such as in the case of the environment support.

Finally, there are also projects supporting environment elements for building real agent applications. In the A&A (agents and artifacts) paradigm (Ricci et al. 2007), artifacts complement agents as tools that can be used and shared among agents. A&A introduces a generic model for agent environment interaction mainly for establishing an alternative to the conventional message-based communication means. As A&A represents a special kind of environment we see that it is orthogonal to our proposal. One interesting way of combining the strength of both approaches could be using the A&A middleware CArtAgO for realizing a distributed version of our environment support. In addition, one could also develop a specific space for A&A facilitating the development of such applications.

In summary, our approach builds on previous and related work, mainly on the ASCML and Repast, and simplifies their basic ideas. Additionally, with spaces it introduces a very powerful concept for realizing agent applications that use some kind of environments, being it virtual worlds or connections to the real world.

[4] http://www.simsesam.de/

[5] http://repast.sourceforge.net/

[6] http://ccl.northwestern.edu/netlogo/

Conclusion

This paper tackled the topic of agent applications and concepts for supporting the developer in constructing multi-agent systems. The first proposition is that an explicit description of a multi-agent system is necessary to enable a developer thinking not only in terms of agents but also in terms of all other involved components. Building on previous and related work it has been highlighted that an agent application often consists of agents and some kind of environment, whereby support for building both is advantageous. As environments may be quite different, ranging from simulated 2d or 3d worlds to real application contexts, application descriptions should reflect these requirements and offer a flexible way for defining constituting parts of the application.

As an answer to the aforementioned demands an extensible XML agent application descriptor has been presented. It cleanly separates the model from the runtime level and introduces, besides agents, spaces as a new core concept for specifying applications. Each application can own as many (same or different) spaces as necessary. A space can be interpreted in a very broad sense and can represent an environment artifact, which can be accessed from all agents of the application. As an example the environment space has been introduced. It allows for defining virtual 2d environments and takes over many aspects of simulated worlds, e.g. the domain model, the agent environment interaction and also the visualization.

Finally, as an example application, the MedPAge application has been shown. It demonstrates the usage of the application and environment descriptor for a resource planning problem in hospitals. One main benefit that can be achieved using the presented techniques is a clear separation of concerns between agent and environment responsibilities. Hence, it was possible to reuse the same system for simulating the scheduling algorithms as well as for real field tests. As one strand of related future work we plan to develop further space types. One very interesting approach is the realization of a truly distributed environment space using CArtAgO as underlying technology.

References

Behrens TM, Dix J, Hindriks KV (2009) Towards an Environment Interface Standard for Agent-Oriented Programming, technical report at Clausthal University, http://www.in.tu-clausthal.de/uploads/media/eisproposal.pdf

Braubach L, Pokahr A, Bade D, Krempels KH, Lamersdorf W (2005) Deployment of Distributed Multi-Agent Systems. Proc. of ESAW, Springer, Berlin.

Ferber J, Gutknecht O, Michel F (2003) From Agents to Organizations: an Organizational View of Multi-Agent Systems. Proc. of AOSE IV. Springer, Berlin.

Jennings N (2001) An agent-based approach for building complex software systems. Commun. ACM 44(4): 35-41.

Paulussen TO, Zöller A, Rothlauf F, Heinzl A, Braubach L, Pokahr A, Lamersdorf W (2006) Agent-based Patient Scheduling in Hospitals. Multiagent Engineering - Theory and Applications in Enterprises, Springer, Berlin.

Ricci A, Viroli M, Omicini A (2007) The A&AProgramming Model and Technology for Developing Agent Environments in MAS. ProMAS'07, Springer.

Unland R, Calisti M, Klusch M (2005) Software Agent-Based Applications, Platforms and Development Kits. Birkhäuser, Basel.

Weyns D, Omicini A, Odell J (2007) Environment as a first class abstraction in multiagent systems. AAMAS 14(1): 5-30.

Wooldridge M (2001) An Introduction to MultiAgent Systems. Wiley, New York.

Zöller A, Braubach L, Pokahr A, Rothlauf F, Paulussen TO, Lamersdorf W, Heinzl A (2006) Evaluation of a Multi-Agent System for Hospital Patient Scheduling. ITSSA 1(4): 375-380.