RULE-BASED ORCHESTRATION OF AGENT-SOCIETIES

Karl-Heinz Krempels and Christoph Terwelp
Databases and Information Systems, RWTH Aachen University, Aachen, Germany

Keywords: Agent societies, Orchestration, Rule-based system.

Abstract: Composing heterogeneous agent-based applications is mostly a complex task due to specific requirements of agents and existing dependencies among agents and societies. Resolving such dependency-networks is subject of agent and agent-society deployment and monitoring. The orchestration task covers automatic deployment, configuration, monitoring and reconfiguration of agent-based applications.
Existing approaches provide static mapping of dependencies and constraints of agent and agent-society descriptions. This leads to a high modification effort, which requires very specialised developer's know-how and can be very complex as well as error-prone, not only when distributing agents over several hosts, but also when launching agents locally.
In this paper a reference model of a deployment infrastructure, a description model for agents and agent-societies and a knowledge-based mechanism for the orchestration of agent and agent-societies are presented with the aim to overcome the disadvantages of the considered existing approaches.

1 INTRODUCTION

Multiagent systems (MAS) are composed of autonomous, interacting, more or less intelligent entities. The agent metaphor has proven to be a promising choice for building complex and adaptive software applications, as it addresses key issues for making complexity manageable at conceptual level (Jennings, 2001). Furthermore, agent technology can be seen as a natural successor of the object-oriented paradigm that enriches the world of passive objects with the notion of autonomous actors. Even though agent applications are developed in various domains, most of them are specialised solutions that are deployed in at most one setting. This is caused by the fact that composition and deployment of agent-based applications is not considered by agent-oriented software engineering methods and therefore neither by multiagent system development frameworks. In consequence, a methodology is chosen independently for each application without consideration of the possibility to integrate the development process or making the effort to derive a commonly accepted methodology.

For object-oriented distributed systems the deployment process is specified in systematical guidelines describing mechanisms for all activities concerned herewith. These guidelines ensure that a properly developed distributed application can be packaged into a reusable, maintainable, and configurable piece of software. However, multiagent systems composed of autonomous proactively (inter-)acting entities differ considerably from distributed systems and the issue of appropriate deployment techniques for MAS has not yet been researched further.

The aim of this paper is to specify agent-based applications at a high-level using dependencies to declare which recommended properties have to be fulfilled for the application to start and run properly. In a first step towards this high-level deployment for MAS a reference model for launching, configuration, reconfiguration and monitoring of distributed multiagent applications is proposed which specifies applications by declaring which and how many agent or agent-society instances shall be instantiated at a time and in which order. A generic meta-model for the specification of agent-based applications is described as part of the reference model. It consists of two layers one for the definition of agent types and one for the composition of agent and agent-society instances belonging to a certain agent-based application scenario.

After an overview of the state of the art in section 2 and a description of the identified requirements in section 3, the model is mapped to a knowledge-base providing the basis for the development of the orchestration capability of agents and agent-based application tools in section 4. In section 5 the approach and some conclusion for future work are summarized.

Krempels K. and Terwelp C..
RULE-BASED ORCHESTRATION OF AGENT-SOCIETIES.
DOI: 10.5220/0003113103670371
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 367-371
ISBN: 978-989-8425-41-6

2 BACKGROUND

The process of deployment is well specified for distributed systems and consists of many phases, identified by the *Object Management Group* (OMG) (Object Management Group, 2003).

For agent-based applications, this process is more dynamic and flexible, because the application constituting elements are autonomous agents instead of passive components. Launching multiagent-based applications differs from starting a distributed component-based application. Component-based applications are hierarchical structured and are usually launched using a single starting point, that creates all needed subcomponents. In contrast, agent-based applications consist of a bundle of autonomous actors that are self-dependent after launching. A minimum requirement for the description of agent-based applications at instance level is the specification of agent instances, of agent-society instances and dependencies among them.

2.1 Terminology

Before going into details some terms used in this paper are defined. Configurations can be defined at two levels: Component level and application level (Castaldi, 2004). For agent-based systems, the agent level and the application level can be distinguished. Considering a single agent a distinction can be made between the static implementation and the running instance. When emphasizing this distinction the former is referred to as *agent type* and the latter as *agent instance*. This distinction can also be made on the application level. The term *society type* refers to the static properties of a multiagent application. A society type is a composition of agent types, supplemented with some (e.g. interaction) constraints. A *society instance* refers to the instantiation of a society, and is composed of single agent instances and concrete dependencies between those instances. The model is recursively defined to allow societies to be part of larger societies on type and instance level.

2.2 Existing Approaches

In (Ricordel and Demazeau, 2000) several agent development frameworks are compared with respect to their capability to support the phases: analysis, design, implementation and deployment of software development processes. It shows that only a few frameworks are addressing the issue of deployment. *ZEUS* (Nwana et al., 1999), *AgentFactory* (Collier, 2001), *Agent Builder* (Acronymics, Inc, 2004) and the *Agent-Society Configuration Manager and Launcher (ASCML)* (Bade et al., 2007; Braubach et al., 2005; Lilienthal and Widyadharma, 2006) are positive exceptions. Agent Academy and AgentFactory framework allow to specify agent applications. They offer support to parametrize agent instances from defined agent types. When launching the application, all agent instances are started at once. ZEUS and AgentFactory use human readable starting scripts that contain commands for starting agents. All these approaches suffer from lack of concepts to describe agent-societies and dependencies between agents and societies. The dependencies have to be resolved by the administrator of an agent-based application before the application is launched. The ASCML defines concepts for agent types, agent instances, society types and society instances. Also the definition of dependencies between these entities is possible. But it can only handle six generic dependency types and application based extensions can only be achieved by modification of the ASCML.

Tools like *editION* (dos Santos Ferreira Júnior et al., 2006), a calligraphic interface for managing agents for the ION agent framework, concentrate on supporting the administrator in designing agent-based applications and does not allow reconfiguration at runtime.

3 REQUIREMENTS

From the presented state of the art in deploying multiagent applications a lack of concepts, standards and tools can be identified, in particular for *launching* and dynamic *reconfiguration* of agent-based applications.

In the following desirable features to achieve dynamic reconfiguration and manageable launching are introduced.

A management tool for the deployment of agent-based applications must provide support for the introduced models for agents, societies and dependencies descriptions. Furthermore, it must provide basic services to edit and modify these entities as well as starting and stopping services for agents and agent societies. Additionally, these services must be available remotely to enable the launching of distributed applications. Therefore, issues of security accounting and write management have to be considered (Sloman, 1995). A launch-process management is required by the basic services to ensure that the right agents and agent-societies are launched at the specified nodes at the correct time. This can be achieved by defining dependencies among agent instances of a society and solving these dependencies at launch time to deter-

mine the right launch time and order for every agent.

Monitoring and reconfiguration services have to be offered by a management agent of agent-based applications. In this way the management agent is enabled to control the whole application. Once an application has been started its administrator has to be informed about its state, resulting from the lifecycles of its entities.

A reconfiguration capability is necessary to enable the management agent to adapt the application according to changing demands. E.g. to start more agents in case of unused computing power. By detecting failures and relaunching agents, as well as detecting agents which are not used by an application anymore, the monitoring service can increase the reliability of agent-based applications and save computing power.

To describe and launch agent-based application there is a need to adapt the agent, the society and the dependency concept. In the ASCML approach six generic dependencies for application independent use have been introduced to model specific dependencies for a given application.

For durable applications concepts for extensibility at design- and runtime must be provided. Otherwise a redesign of the application is necessary to change its configuration.

4 APPROACH

This approach for distributed deployment of multiagent applications is based on the idea of specialised service agents that are responsible for launching and managing agents and societies on their own platform. These service agents are called IASCML (Intelligent Agent-Society Configuration Management and Launcher).

4.1 Deployment Reference Model

The deployment reference model used here is similar to the model of the ASCML agent (Braubach et al., 2005) and assumes a running IASCML agent on each platform. Each IASCML agent controls an agent repository containing archives with agent and agent-society applications. The archives contain description files for each launchable agent included. These description files simplify the launching and configuration task as they enable developers to map special agent configuration parameters to well defined parameters of the agent description model. A welcome effect of this is that we hide special agent configurations from the user of the agent. This also ensures that an agent archive can be used in the same way on all the agent platforms running an IASCML agent.

Agent-societies can be handled similarly since they are build on top of well defined agent descriptions in this model.

4.2 XML-Model-Description

A uniform description for agents and agent-societies is ensured by concepts based on the ASCML XML-schema (Braubach et al., 2005). The concept states/assumes that agent description files end with the suffix **.agent.xml* and agent-society description files with **.society.xml*. This enables the IASCML agent to identify agent and agent-society descriptions in all the archives from its repository.

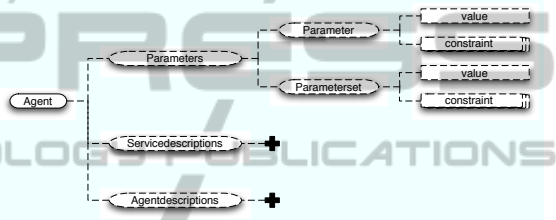

Figure 1: XML agent description model.

Figure 1 depicts the structure of an agent type specification. The root agent tag captures important properties of an agent such as the agent type, the implementation class, and the platform type. Within the *servicedescriptions*- and *agentdescriptions*-tags FIPA-compliant descriptions for the agent itself and the services offered by the agent can be declared. The *parameter*-tag encloses single-valued parameters and multi-valued parameter sets that are passed as *key=value*-arguments to the main class of the agent at creation time. Parameters specified by the agent type can be declared mandatory or optional. Parameter values may be provided as default settings. All parameters specified for an agent type are inherited by corresponding agent instances. The agent instances have to provide parameter values for all mandatory parameters for which the agent type provides no values. Predefined values of agent type can also be overwritten with new values.

Besides the agent types meta-model there is also a meta-model for society types defined.

Figure 2 depicts the XML root-element of a society type followed by a set of child tags. The *imports*-, *agenttypes*- and *societytypes*-tags specify lists of referenced elements. The *societyinstances*-tag is the only mandatory tag. Within this tag one or more society instances representing different application settings can be listed.

Figure 2: XML society description model.

The elements constituting a society instance are shown in Figure 3. A set of agent instances which have to be created on the local agent platform on start of the society instance can be specified. Every agent instance can optionally be provided with parameter values. In addition to that a set of agent platform dependent tool options can be specified. These tool options are used to start special tool agents (e.g. sniffer or logger) at once with the agent instances and to register an agent automatically with the selected tool agent.

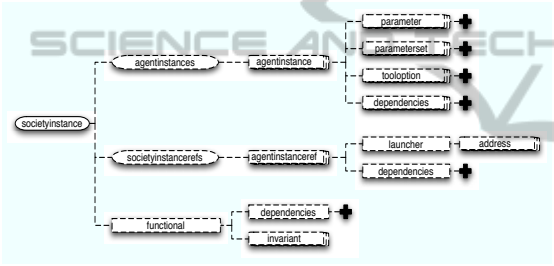

Figure 3: XML society instance description model.

Besides the agent instances a society instance can contain an arbitrary number of subsocieties which again can contain further subsocieties. This allows a recursive definition of applications and facilitates the creation of distributed multiagent applications. Each referenced subsociety instance refers to a concrete society instance, which itself belongs to a declared society specification. For the purpose of starting a remote society, a so-called launcher identifier can be declared. This identifier designates the remote IASCML agent responsible for starting the corresponding remote society instance. At last a society instance can be declared when to be functional by specifying a set of dependencies and invariants that have to be fulfilled. The IASCML supervises these dependencies and invariants at runtime and for dependencies being marked as active it engages autonomously in restarting the non-functional parts.

The last focus is on the meta-model for dependencies (Braubach et al., 2005). As shown in the society instance scheme above, dependencies can be defined for agent instances, society instances, and to describe the functional state. Figure 4 depicts the different

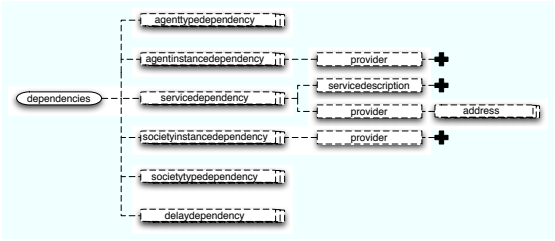

Figure 4: XML dependency description model.

types of dependencies. An agent type dependency can e.g. be used to wait for an arbitrary number of agents of a specified type to be running, while an agent instance dependency exactly refers to one designated agent, identified by its unique agent id. Both kinds of dependencies also exist for the society element.

4.3 Knowledge-based Model Description Processing

Processing the agent and society descriptions by an ASCML agent as proposed in (Braubach et al., 2005) suffers from one main disadvantage, the processing of static description modes. This restricts a developer to specifying the configuration of an agent or agent-society exclusively at design time.

To overcome this disadvantage a knowledge-based processing of agent and society descriptions is proposed in this approach. This provides the possibility to introduce additional dependency types for agents at runtime, and to define them at a finer granularity. E.g., to use a description property of an agent, a service description property or even a boolean expression on any of the above as a dependency target.

Therefore, a rule-based engine is used providing a knowledge base and a declarative programming language operating on it. For a rule-based deployment and management of agents and agent societies a mapping of the introduced XML-model descriptions to a knowledge-based representation is necessary, capturing the concepts described above.

To orchestrate a society based on its description model from the knowledge base all the dependencies are mapped to rules. Additionally rules are defined for each society to control its lifecycle.

Dependencies for remote agents or subsocieties are mapped to special rules producing an interaction among IASCML agents (when they are activated) requesting the launch of an agent or subsociety.

5 CONCLUSIONS & OUTLOOK

In this paper an extension of the ASCML approach from (Bade et al., 2007; Braubach et al., 2005) is discussed. The extension provides runtime adaptivity for agent-society structures and more accurate modelling possibilities.

The IASCML approach was implemented with the help of the rule-based system *Jamocha* that provides the required modelling and processing capabilities for knowledge-based models. The approach was tested in a lab showing that the orchestration of agents and agent-societies is improved in comparison to the ASCML approach.

Feedback from ASCML users regarding the usability of the ASCML's GUI, in particular the creation and modification process of agent and agent-society description files has been received. This motivates the redesign of the ASCML GUI and its functionality. The intention is to move the creation and modification task of agent and agent-society descriptions to a new developed plugin for Protégé (Gennari et al., 2002). The usability of the new IASCML GUI needs to reworked to provide for a convenient way of handling distributed agent-based applications.

The specification of agent-platforms is not supported by the IASCML yet. This is would very useful to define dependencies between agents and platforms and to extend the ASCML agent with load balancing capabilities among network nodes. Thus it will subject of further research and development.

To provide the IASCML agent to the agent community we will contribute the code to the JADE (JADE, 2010) developer team. Furthermore, we are going to contribute the agent and agent-society description models to FIPA with the aim to include it in a FIPA specification document.

Our longterm vision is to provide an orchestration tool for agent-based applications and to simplify the deployment and orchestration process of agents and agent-societies. This will help agent-based applications to become (re)usable to a broad community.

REFERENCES

Acronymics, Inc (2004). Agentbuilder - user guide v1.4. http://www.agentbuilder.com/.

Bade, D., Lilienthal, S., Krempels, K.-H., and Widyadharma, A. S. M. (2007). Agent-society configuration manager and launcher. In Bellifemine, F., Caire, G., and Greenwood, D., editors, *Developing Multi-Agent Systems with JADE*, pages 207–223, West Sussex, England. John Wiley & Sons, Ltd.

Braubach, L., Pokahr, A., Bade, D., Krempels, K.-H., and Lamersdorf, W. (2005). Deployment of distributed multi-agent systems. In Marie-Pierre Gleizes, Andrea Omicini, F. Z., editor, *ESAW 2005*, pages 261–276. Springer-Verlag GmbH.

Castaldi, M. (2004). *Dynamic Reconfiguration of Component Based Applications*. PhD thesis, Department of Computer Science, University of L'Aquila, Italy.

Collier, R. W. (2001). *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications*. PhD thesis, University College Dublin.

dos Santos Ferreira Júnior, A. M., Vala, M., Pereira, J. A. M., Jorge, J. A. P., and Paiva, A. (2006). A calligraphic interface for managing agents. *WSCG 2006*, pages 25–31.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubzy, M., Eriksson, H., Noy, N. F., and Tu, S. W. (2002). *The Evolution of Protégé: An Environment for Knowledge-Based Systems Development*.

JADE (2010). The JADE Project Home Page. http://jade.tilab.com/.

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

Lilienthal, S. and Widyadharma, S. (2006). Agent society configuration manager and launcher. In *Informatiktage 2006*, pages 1–4.

Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). Zeus: a toolkit and approach for building distributed multi-agent systems. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 360–361, New York, NY, USA. ACM Press.

Object Management Group (2003). Deployment and configuration of component-based distributed applications specification.

Ricordel, P.-M. and Demazeau, Y. (2000). From analysis to deployment: A multi-agent platform survey. In *ESAW 2000*, pages 93–105, London, UK. Springer-Verlag GmbH.

Sloman, M. (1995). Management issues for distributed services. In *Proc. of the 2nd Int. Workshop on Services in Distributed and Networked Environments*, pages 52–55. IEEE.