Ontology-Based Flexible Multi Agent Systems Design and Deployment for Vertical Enterprise Integration

Christos Alexakos[1], Manos Georgoudakis[2], Athanasios P. Kalogeras[2], and Spiridon L. Likothanassis[1]

[1] Dept of Computer Engineering and Informatics, University of Patras, Greece
{alexakos,likothan}@ceid.upatras.gr
[2] Industrial System Institute, Greece
{kalogeras,georgoud}@isi.gr

Abstract. Empowering autonomic control in the enterprise environment highly contributes in the quest for a higher level of flexibility. Multi-Agent Systems (MAS) may be utilized to this end along with the enterprise environment model leading to a decentralization of the manufacturing production processes. The current work proposes a framework along with the necessary software tools for the modeling of MAS through ontologies, its design and deployment in the enterprise / manufacturing environment.

Keywords: ontologies, multi agent systems, vertical enterprise integration.

1 Introduction

The need to increase the competitiveness of enterprises, especially in the manufacturing sector, is profound. Intra-enterprise interoperability is a prerequisite to this end, making possible the vertical integration of systems / applications residing at different levels of the classical manufacturing environment hierarchy. This need is especially felt when modern enterprises have to support such advanced business models as mass customization that require a robust enough environment that may react effectively and adapt to unexpected events and uncertain timing. Distributing intelligence and providing autonomy to different levels in the manufacturing environment hierarchy may positively contribute to this end. MAS along with a model representation of the enterprise environment and the relevant semantics are fundamentals for such an approach.

The proposed framework is based on the concept of simple definition of the functionalities of an integration MAS that permits system semi-automatic implementation and deployment, with the following challenges:

- The description of MAS modules, their functionalities, processes and data exchange protocols must follow a human understandable way. Model-based design using UML is a good paradigm for design of systems behavior by engineers.
- The implementation phase of MAS must include the least possible code implementation, accelerating the deployment procedure and enforcing system re-usability. The use of modular system architecture, where the modules can be easily developed and re-used, in combination with code-generation of system main components can significantly decrease the need for code programming.

- System definition must be computer-understandable, allowing computer software to read MAS description and proceed to the appropriate actions, such as code generation, agent creation and data mapping. XML and RDF are open standards that can easily define concepts, be delivered to systems and parsed by them.
- The information integration between heterogeneous systems is a major issue in enterprise system integration; without the "common understanding" of the data exchanged between systems the orchestration of the business processes is meaningless.

In order to face the aforementioned challenges, the proposed framework is accompanied by a concrete methodology for MAS definition, implementation and deployment phases based on GAIA [1] agent-oriented software engineering (AOSE) methodology. Furthermore, a set of tools support the three phases of the methodology establishing an integrated architecture for the framework.

The rest of the paper is organized as follows: In section 2 a brief presentation of the related works on ontology-driven design and deployment of MAS is provided. Section 3 presents the proposed framework methodology and its main components along with its supporting tools. Finally, some conclusion is given in chapter 4.

2 Related Work

MAS model development is supported by a number of AOSE methodologies and technologies [2], characterized by different levels of maturity. GAIA methodology is very popular [3] because it provides a strong design tool for engineers in order to describe the functionalities of a MAS along with the behavior of the agents acting in it. According to the GAIA-approach the behavior and interaction of an agent-based system is described by a set of roles with role related activities and a set of interactions among the roles. GAIA is used in various approaches for MAS design in enterprise / manufacturing integration [4].

During the recent years ontologies have been used as tools in model – driven architectures for defining the detailed functionality of a MAS. The semantic basis is the frame-based ontology ONTODM. ONTOMADEM [5] is a knowledge-based tool for designing MAS based on the MADEM MAS design methodology. ONTOMADEM is using a frame-based ontology constructed with Protégé tool. Bittencourt et al. [6] propose an ontological model for defining MAS functionalities and components driven for two ontologies, one describing GAIA methodology agents and one describing JADE agent implementation. The model uses SWRL rules for mapping GAIA agent roles and activities to JADE agents and behaviors. Nevertheless, a common denominator in most of these approaches is the lack of automatic creation of MAS.

In the runtime phase of MAS, ontologies in most approaches are used for supporting semantics common understanding of the integrated data exchanged between agents. Few proposals elaborate ontologies from the MAS design phase to deployment. O-PRS (Ontology driven Procedural Reasoning System like model) [7] utilizes OWL ontologies to express the concepts in Believe, Desire, Intention (BDI) agent architecture. The ontologies add the appropriate semantics to the exchanged messages in order for the agents to understand how to act. Nyulas et al. [8] proposed an ontology-driven framework for design and deployment of MAS on top of Java Agent DEvelopment Framework (JADE) framework (http://jade.tilab.com/). The framework is based on the definition of MAS functionality using three Ontology Web Language (OWL) ontologies (Data Source, Task-Method and Deployment). For MAS deployment, special purpose agents (Controller Agent and Configurator Agent) are used in order to create and monitor task agents according to the semantic configuration.

3 Proposed Integration Framework

3.1 Design to Deployment Methodology

The proposed methodology covers the three phases of system engineering: design, implementation and deployment. For simplicity, the testing phase is considered as part of the implementation phase. Figure 1 depicts the major and optional steps of the methodology in the three phases.

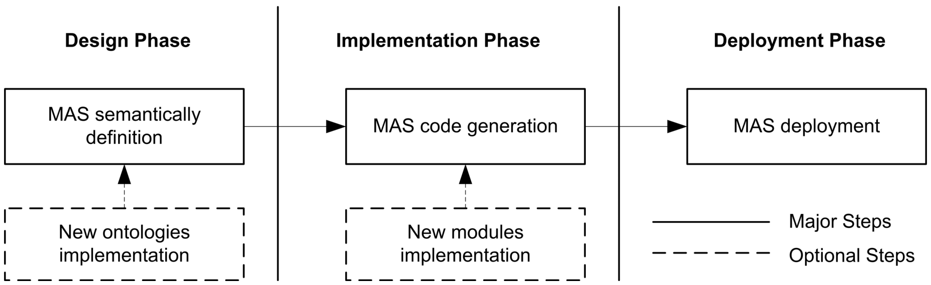

Fig. 1. Design to Deployment Methodology

The core aspect of the proposed methodology is the MAS semantic definition in terms of ontologies. Ontologies represent powerful tools for defining the knowledge of a domain. In the case of enterprise system integration knowledge includes the business processes, the exchanged information and business logic executed from the participating systems. Ontologies describe the concepts in terms of axioms that for the shake of simplicity can be considered as sentences such as "Agent A executes the Invoice Procedure". In modern ontology languages like OWL, a more object oriented approach is used where entities follow a hierarchical structure enforced by relationships between them. OWL is an ontology language generated on the concept of Semantic Web, thus its main representation format is based on XML and RDF, permitting both human and computer understandability. For this reason, it is used as the basis for MAS definition in the proposed framework.

OWL Ontologies follow the T-Box and A-Box distinction which is drawn in Description Logics. OWL ontologies are composed of two functional parts, the first part is the ontology scheme (T-Box) and the second is a set of instances of the ontology scheme containing the data related to the described domain (A-Box). The proposed methodology uses as core ontology scheme the OWL-GMO (Gaia MAS Ontology)

ontology that defines the core entities of MAS system following the widely accepted GAIA design methodology. Each agent role is depicted as an instance of OWL-GMO. Furthermore, OWL-GMO provides entities and relations for the definition of agent behaviors and messages exchanged. Other pre-existing ontologies or newly-composed ones can be integrated in the core ontology scheme in order to define the information semantics of the data used by the participating systems and agents.

The implementation phase includes the code generation of the components defined by OWL-GMO followed by the packaging of external modules that execute specific business logic functionalities. Agents and their functionalities (behaviors and message exchanged) are implemented following the specification of agent implementation of JADE. JADE supports the implementation of distributed software agent systems where software agents are running in different hosts. Moreover, the agent instances are managed according to globally accepted FIPA specifications. Finally, JADE agents follow Agent Communication Language (ACL), a well-structured schema for message exchange.

The specific business logic or functionalities that have to be executed by the agents must be implemented as modules providing specific APIs for invocation by the core agent implementation. In this case, the binaries or java code of these modules will be included in the core generated code. Finally, the code will be compiled and ready for deployment to the real enterprise environment.

3.2 Architecture Components

Each distinct step of the aforementioned methodology is supported by software tools either providing graphical user interface to the users or automatically executing tasks. Figure 2 depicts the system components of framework methodology software realization. The architectural components are:

- *Ontology Editor*. It is GUI for composing and instantiating the ontologies used in the concept of the framework. Protégé tool (http://protege.stanford.edu/) is used for this purpose.
- *Ontology Importer*. It is a module that manages the OWL ontology files composed by the Ontology Editor for a specific project (MAS definition). Furthermore, it keeps versions of the ontologies for potential use.
- *Code Generator* is the component responsible for the transformation of the MAS definition to code running on top of JADE MAS platform. The Code Generator uses an OWL Ontology Reasoner for the conceptualization –i.e. the identification of hierarchy and relations - of the imported ontologies. JENA semantic framework (http:// jena.apache.org) with Pellet (http://clarkparsia.com/pellet) reasoning support is used for this purpose.
- External Module Importer is a GUI responsible for collecting the external modules (binaries or code) needed for MAS implementation.
- Packaging Manager is responsible for managing the generated code and the imported modules and creating the final application package for compilation.

- Binary Builder is the tool that builds the final MAS system. Binary Builder can be directly connected to the deployment or testing environment in order to automatically deploy the binary code.

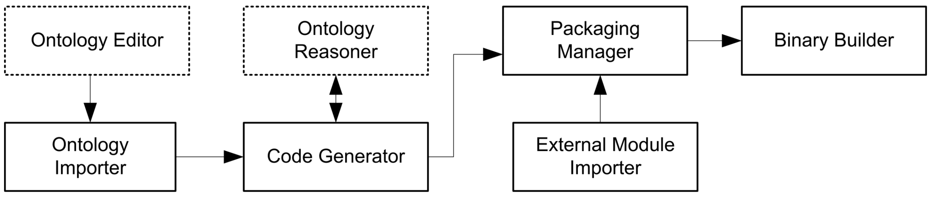

Fig. 2. Framework Methodology Software Realization Architectural Components

3.3 OWL-GMO

As aforementioned, OWL-GMO is the core ontology of the proposed framework used to define the agents and their behavior in MAS. OWL-GMO entities come from the terms defined in the GAIA methodology. The ontology is based on two super-classes:

- *AgentSystemEntity* which abstractly defines the entities used by GAIA methodology to define a MAS.
- *AbstractConcept* which defines supporting entities used to define concepts of the behavior and functionality of the MAS (data, processes, APIs, etc).

According to GAIA, each agent system comprises a set of agent roles. Each role is defined by four attributes: responsibilities, permissions, activities, and protocols. Responsibilities determine functionality and, as such, are perhaps the key attributes associated with a role. Responsibilities are divided into two types: liveliness properties and safety properties. In order to realize responsibilities, a role has a set of permissions. Permissions are the rights associated with a role. Therefore, the permissions of a role identify the resources that are available for that role in order to realize its responsibilities. In the kind of system that has been typically modeled in this work, permissions tend to be information resources. Activities are actions associated with the role and are carried out by the agent without interacting with other agents. Finally, a role is also identified with a number of protocols which define the way that it can interact with other roles. Each protocol is defined by its initiator role, the responder roles, input data, output data and the process executed.

Figure 3 depicts the OWL-GMO visualization. The classes *AgentRole, RoleResponsibility, RolePermission, RoleProtocol* and *RoleActivity* are the subclasses of the class *AgentSystemEntity* and define the main entities of GAIA methodology. For *RoleResponsibility* there are two subclasses, *LivenessProperty* and *SafetyProperty* following GAIA specifications. These classes following by relationships expressing specific concepts and restrictions (i.e. *AgentRole accessResourcesAccordingTo RolePermission*) are used for the definition of the MAS.

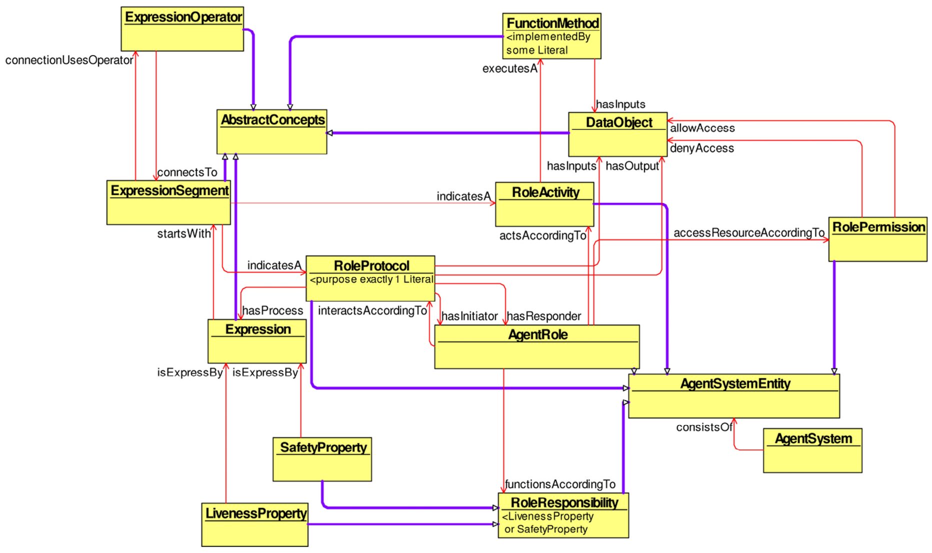

Fig. 3. OWL-GMO Ontology

Furthermore, OWL-GMO uses the subclasses *DataObject, FunctionMethod, Expression, ExpressionSegment* and *ExpressionOperator* to express additional concepts reading data, operations and processes. *DataObject* class is used to define the used data, usually enriched with external ontologies describing the information data used for a specific project. *Expression* denotes a concept that consists of *ExpressionSegments* which indicate either an activity or a protocol. *ExpressionSegments* are connected via *ExpressionOperator* in order to compose a flow of actions that define the activity. *FunctionMethod* is designed in order to be instantiated as a method from software libraries. It is mapped to the external modules API methods for executing specific business logic functionalities.

3.4 MAS Implementation

The outcome of the design phase is the instantiated OWL-GMO where individuals are depicted by the agents and their functionalities. At the next step, being MAS implementation, the relative code of the agents will be generated according to JADE agent framework guidelines. In order to generate code, sets of "transformation" scripts are executed, which logic follows the main principles presented by Spanoudakis and Moraitis [9].

In order to manage the data used by the agent system and depicted in the imported ontologies extending Data Object, the function *createDataObjects*() generates java classes based on the heredity feature which is supported by both OWL and Object oriented program languages. The *create_literal_var()* function creates a class public property of java common data type (String, int, etc) associated to DataProperty range type. The *create_object_var()* function creates a public property of the corresponding java class to ObjectProperty range ontology class. Java cardinality is defined by an array if there are multiple values. This transformation allows Java classes to be used in the generated code associated with the MAS definition.

```
Function createDataObjects():
foreach (OWL Class) C
  if (OWL Class) C is subclass of (OWL Class) C1
    create_java_class_extends(C.name,C1.name)
  else
    create_java_class(C.name)
  endif
  for each (OWL DataTypeProperty) DP of (OWL Class) C
    cardinality = getPropertyCardinality(DP)
    create_literal_var(DP.name,C.name,cardinality)
  endforeach
  for each (OWL ObjectProperty) OP of (OWL Class) C
    cardinality = getPropertyCardinality(OP)
    create_object_var(OP.name,C.name,cardinality)
  endforeach
endforeach
```

Having generated the Java data object classes, the next step is the creation of the agents. In JADE framework agents are realized as extensions of *jade.core.Agent and their* tasks (activities or protocols) are implemented as a JAVA classes extending the *jade.core.behaviours.Behaviour* class. The received / sent messages for each agent are implemented using the *jade.lang.acl.ACLMessage* class. Function *createAgents()* depicts the code generation script for agents and their behaviors.

```
Function createAgents():
foreach (AgentRole as Agent)
  create_Agent(Agent)
  foreach (AgentActivity(Agent actsAccordingto)as
    Activity)
    Behaviour = create_Behaviour(Activity)
    attach_Behaviour_to_Agent(Behaviour,Agent)
  endforeach
  foreach (RolePermission(Agent interactsAccordingTo)as
    Permission)
    DataObject = getDataObject_isAllowed(RolePermission)
    attach_Data_Object_to_Agent(DataObject,Agent)
  endforeach
  foreach (AgentProtocol(Agent hasInitiator ||
    interactsAccordingTo|| hasResponder)as Protocol)
    Expression = getExpression_hasProcess(Protocol)
    attach_protocol_behaviour(Expression,Agent)
  endforeach
endforeach
```

For the agent protocols that are released by *attach_protocol_behaviour()* function, the corresponding behaviors and ACL messages are generated according to the process flow defined by the related Expression. In each transformation the input and output data of behaviors are mapped according to Java data objects created by the function *createDataObjects()*.

The next step is the gathering of all the external modules/java classes and adding them to the project classpath. The final step comes with the compiling and building of agent executable code that will run on top of an installed JADE platform at the enterprise environment.

4 Conclusion

The paper presents a framework relevant to the modeling, design, implementation and deployment of a MAS for the enterprise environment allowing the increase of its flexibility and decision making autonomy. In this context production process may be easier and more effectively decentralized. The different software tools are presented that make it possible to model the MAS in terms of ontologies, implement it and deploy it to the enterprise environment through Java code generation.

References

1. Zambonelli, F., et al.: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology 12(3), 317–370 (2003)
2. Akbari, O.Z.: A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance. International Journal of Computer Engineering Research 1(2), 14–28 (2010)
3. Leitão, P., Vrba, P.: Recent Developments and Future Trends of Industrial Agents. In: Mařík, V., Vrba, P., Leitão, P. (eds.) HoloMAS 2011. LNCS, vol. 6867, pp. 15–28. Springer, Heidelberg (2011)
4. Girardi, R., Leite, A.: A knowledge-based tool for multi-agent domain engineering. Know.-Based Syst. 21(7), 604–661 (2008)
5. Bratukhin, A., Sauter, T.: Functional Analysis of Manufacturing Execution System Distribution. IEEE Transactions on Industrial Informatics 7(4), 740–749 (2011)
6. Bittencourt, I.I., Bispo, P., Costa, E., Pedro, J., Véras, D., Dermeval, D., Pacca, H.: Modeling JADE Agents from GAIA Methodology under the Perspective of Semantic Web. In: Filipe, J., Cordeiro, J. (eds.) ICEIS 2009. LNBIP, vol. 24, pp. 780–789. Springer, Heidelberg (2009)
7. Mousavi, A., Nordin, M., Othma, Z.A.: An Ontology Driven, Procedural Reasoning System-Like Agent Model, For Multi-Agent Based Mobile Workforce Brokering System. Journal of Computer Science 6, 557–565 (2010)
8. Nyulas, C., et al.: An Ontology-Driven Framework for Deploying JADE Agent Systems. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 573–577. IEEE Press, New York (2008)
9. Spanoudakis, N., Moraitis, P.: Gaia Agents Implementation through Models Transformation. In: Yang, J.-J., Yokoo, M., Ito, T., Jin, Z., Scerri, P. (eds.) PRIMA 2009. LNCS, vol. 5925, pp. 127–142. Springer, Heidelberg (2009)