Engineering Applications of Artificial Intelligence 25 (2012) 159–173

Contents lists available at SciVerse ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Model-driven engineering techniques for the development of multi-agent systems

José M. Gascueña [a], Elena Navarro [a,b], Antonio Fernández-Caballero [a,b,*]

[a] Instituto de Investigación en Informática de Albacete (I3A), 02071 Albacete, Spain
[b] Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, 02071 Albacete, Spain

ARTICLE INFO

Article history:
Received 8 June 2010
Received in revised form
4 July 2011
Accepted 22 August 2011
Available online 9 September 2011

Keywords:
Agent-based method
Model-driven development
Meta-modeling
MDE-MAS method and tool
Agent-oriented software development
Multi-agent systems
Surveillance systems
Eclipse-Modelling Framework
Graphical Modelling Framework

ABSTRACT

Model-driven engineering (MDE), implicitly based upon meta-model principles, is gaining more and more attention in software systems due to its inherent benefits. Its use normally improves the quality of the developed systems in terms of productivity, portability, inter-operability and maintenance. Therefore, its exploitation for the development of multi-agent systems (MAS) emerges in a natural way. In this paper, agent-oriented software development (AOSD) and MDE paradigms are fully integrated for the development of MAS. Meta-modeling techniques are explicitly used to speed up several phases of the process. The Prometheus methodology is used for the purpose of validating the proposal. The meta-object facility (MOF) architecture is used as a guideline for developing a MAS editor according to the language provided by Prometheus methodology. Firstly, an Ecore meta-model for Prometheus language is developed. Ecore is a powerful tool for designing model-driven architectures (MDA). Next, facilities provided by the Graphical Modeling Framework (GMF) are used to generate the graphical editor. It offers support to develop agent models conform to the meta-model specified. Afterwards, it is also described how an agent code generator can be developed. In this way, code is automatically generated using as input the model specified with the graphical editor. A case of study validates the method put in practice for the development of a multi-agent surveillance system.

1. Introduction

Currently, the use of the model-driven engineering (MDE) approach throughout the software development process is gaining more and more attention (Gasevic et al., 2009). MDE concerns the exploitation of models as the cornerstone of the software development process. It allows both developers and stakeholders to use abstractions closer to the domain than to computing concepts. Thus, it reduces the complexity and improves the communication. As the main aim of MDE is to develop software, this paradigm uses software models as their expression vehicle.

Sometimes, models are constructed to a certain level of detail, and then code is written by hand in a separate step. Some other times (most often) code is automatically generated from the models, ranging from code skeletons to completely deployable products. Usually, these models are specified by instantiating *meta-models*, that is, models to describe models. The basic idea of meta-model is to identify the general concepts in a given problem domain and the relations used to describe models. This serves as a strategy that forces a clear distinction between the real problem to be solved by the system and the framework where the model lives.

The use of MDE has the following consequences for a software development process. (1) More time can be devoted to analyzing and designing models. (2) The time necessary to perform coding tasks is reduced, as code generators are usually available to carry them out in an automatic way. The programmers are responsible for completing those parts of the system that developers either have decided not to generate or cannot do. (3) The quality of the developed system is improved, as the generated code (usually) does not have bugs. And, (4) productivity is improved as the time necessary for coding is reduced. More effort is devoted to solve errors during early phases of the life cycle, avoiding in this way the "snow ball" effect (Pressman, 2010). Moreover, MDE provides inter-operability among heterogeneous systems thanks to the specification of bridges between different technologies. Portability is also improved to adopt a new technology, just developing a new code generator, as the models are independent of any technology. In summary, MDE offers important benefits in aspects as important as productivity, portability, inter-operability and maintenance (Kleppe et al., 2003).

In contrast, MDE also demonstrates some drawbacks (Mattsson et al., 2009). Although MDE automates the steps from detailed design to implementation, as described before, at present

* Corresponding author at: Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, 02071 Albacete, Spain. Tel.: +34 967 599200; fax: +34 967 599224.
E-mail address: Antonio.Fdez@uclm.es (A. Fernández-Caballero).

0952-1976/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2011.08.008

it is not able to automate enforcement of the architecture on the detailed design. This is due to the inability to model architectural design rules. Unfortunately, this is a bottleneck in large MDE projects, as the developers have to carry out this task manually, that is, in a similar way to more traditional approaches.

On the other hand, multi-agent systems (MAS) are appropriate to model and develop complex software applications with high need of autonomy, communication among autonomous elements and distribution (Jennings et al., 1993; Karageorgos et al., 2003; Posadas et al., 2008; Leitão, 2009). More and more, MAS are introduced in different domains (e.g. teleoperated systems, Rodríguez-Seda et al., 2010, intruder detection systems, Jha and Massan, 2002, and so on). Agent-oriented software development (AOSD) (Henderson-Sellers and Giorgini, 2005) is the paradigm described for the construction of this kind of systems. Lately, several methodologies, such as Gaia (Wooldridge et al., 2000), MaSE (Deloach et al., 2001), ADELFE (Bernon et al., 2003), Prometheus (Padgham and Winikoff, 2004), Tropos (Bresciani et al., 2004), and INGENIAS (Pavón et al., 2006) have come up following this paradigm. Every one of them exhibits the characteristics that a software methodology (Bauer and Odell, 2005) should have, that is, a *modeling language* and a *software process*. A modeling language is used for the specification of the corresponding models by using its specific syntax (notation) and its associated semantics. A software process specifies the development activities, the inter-relationships among them, and how they are performed. In the definition of the AOSD methodologies two different approaches have been followed.

In first place, some of them, such as ADELFE, extend a generic modeling language – Unified Modeling Language (UML) (Rumbaugh et al., 2004) – and a process described in the context of software engineering – Unified Software Development Process (Jacobson et al., 1999). Other approaches, such as Prometheus, have their own language and development process. However, no matter which approach is followed by a methodology, there are always compelling arguments to provide tool support for their application. This is why, the MDE approach for building supporting tools emerges naturally as a way to improve the development of agent-based software applications. This is the main argument that has conducted this work, namely, to show how the AOSD paradigm can be integrated with the MDE approach.

The rest of the paper is organized as follows. In Section 2 some previous works related to our proposal are revisited. Then, in Section 3 our new proposal of the use of model-driven engineering in Prometheus methodology are described. In Section 4 the Prometheus Model Editor is introduced in detail through describing the meta-model definition, the graphical editor construction and the code generation. A case of study related to the development of a multi-agent surveillance system is used in Section 5 to validate the method. Lastly, Section 6 offers some conclusions and hints towards future work.

2. Related works in model-driven engineering for multi-agent systems

Meta-models define general concepts of a given problem domain and their relationships. General concepts and relationships are really a language that, for instance, may be used to specify the domain's requirements (Smolík, 2006). The advantage of introducing meta-models in the development process is the higher abstraction level to work with.

According to Molesini (2008), meta-models should be used in AOSD as they describe each methodology and infrastructure in a compact and precise way. Moreover, they form the basis for analyzing and comparing methodology and infrastructure. Indeed, the elements and the relationships that describe them are present in the meta-models. Besides, they help to study the existing gap between agent-oriented methodologies and agent-oriented infrastructures, as they allow one to isolate the main concepts of the system from the underlying technology. Finally, meta-models are the starting point to define methodologies along with their corresponding agent-oriented infrastructures.

Unfortunately, there are several agent-oriented modeling languages but not a standard one. In principle UML could be considered as the standard to be used, but it is not the best tool for modeling agent-based systems (Bauer, 2001). This is basically due to two reasons: (1) compared to objects, agents are active as they take initiative and have control over external requests; and (2) agents do not only act in isolation but in cooperation or coordination with other agents. Several languages that extend UML have been proposed so far to solve this problem. For instance, Agent UML (AUML) (Bauer et al., 2001) is the first agent modeling language that follows this approach. It provides interaction protocol diagrams and agent class diagrams as extensions of UML's sequence and class diagram, respectively, as a solution to the stated problems. However, the absence of a meta-model and modeling tools are the main drawbacks that explain why this language is not widely accepted. More recently, the Agent Modeling Language (AML) (Cervenka and Trencansky, 2007), a semi-formal visual modeling language based on the UML 2.0 superstructure has been proposed. It is supported by tools like (Enterprise Architect, 2010; StarUML, 2010). However, AML does not concern about code generation, as it focuses its attention on specification tasks. Despite the main aim of AML is to offer a new and well documented unified language suitable for industrial development, unfortunately, most research groups are still using classic methodologies (Henderson-Sellers and Giorgini, 2005), such as INGENIAS or Prometheus, to carry out the modeling of agent-based applications.

Now let us focus on one of the principal agent-oriented methodologies, namely INGENIAS (Pavón et al., 2006). The basis of INGENIAS methodology is the definition of a MAS meta-model described by using GOPRR (Graph, Object, Property, Relationship, and Role) (Kelly et al., 1996). A set of agent-oriented MDE tools (model edition, verification, validation and transformation) are integrated into the INGENIAS Development Kit (IDK) (Gómez-Sanz et al., 2008). The INGENIAS meta-model describes the elements for modeling MAS from different perspectives—agent, organization, environment, goals and tasks, and interaction (Fuentes-Fernández et al., 2010). The agent perspective focuses on the elements necessary to specify the behavior of each agent. The organization perspective shows the system architecture. From a structural point of view, the organization is a set of entities with aggregation and inheritance relationships used to define a schema where agents, resources, tasks and goals exist. Under this perspective, groups may be used to decompose the organization, plans, and workflows to establish the way the resources are assigned, whose tasks are necessary to achieve a goal, and who has the responsibility for carrying them out. The environment perspective defines the agents' sensors and actuators, and identifies the system resources and applications. The goals and tasks perspective describes the relations between tasks and goals. The interaction perspective describes how the coordination among agents is performed. The IDK tool supports the INGENIAS methodology, so that each one of the previous concepts are specified using either an UML-like or INGENIAS specific notation. This facility allows users familiar with the UML notation to reduce the learning curve of INGENIAS. Moreover, the IDK tool has a module for JADE code generation, and a mechanism to define templates used to develop code generation modules for the required target platform.