Computer Standards & Interfaces 43 (2016) 91–115

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

Declarative specifications for the development of multi-agent systems

Moharram Challenger [a,*], Marjan Mernik [b], Geylani Kardas [a], Tomaž Kosar [b]

[a] International Computer Institute, Ege University, Izmir, Turkey
[b] Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia

ARTICLE INFO

Article history:
Received 1 October 2013
Received in revised form 26 August 2015
Accepted 29 August 2015
Available online 4 September 2015

Keywords:
Domain-specific language
Multi-agent system
Semantic Web
Formal semantics
Declarative specifications

ABSTRACT

The designing and implementation of a multi-agent system (MAS), where autonomous agents collaborate with other agents for solving problems, constitute complex tasks that may become even harder when agents work in new interactive environments such as the Semantic Web. In order to deal with the complexities of designing and implementing a MAS, a domain-specific language (DSL) can be employed inside the MAS's development cycle. In such a manner, a MAS can be completely specified by programs written in a DSL. Such programs are declarative, expressive, and at the right abstraction level. In this way the complexity of MAS development is then partially shifted to DSL development and the task herein can be much more feasible by using a proper DSL development methodology and related tools. This paper presents and discusses our methodology for DSL development based on declarative formal specifications that are easy to compose, and its usage during MAS development. A practical case-study is also provided covering an example of a MAS's development for expert finding systems. By using denotational semantics for precisely defining the language, we show that it is possible to generate the language automatically. In addition, using attribute grammars makes it possible to have modular methodology within which evolutionary language development becomes easier.

1. Introduction

A software agent is an encapsulated software system situated within a certain environment, and is capable of flexible autonomous action within this environment in order to meet its design objectives [1]. These autonomous, reactive, and proactive agents can also behave in a cooperative manner and collaborate with other agents for solving common problems. In this way, these intelligent agents constitute systems called multiagent systems (MASs).

The design and implementations of MASs are complex tasks when considering their dynamicity and autonomous characteristics. Their behaviour is more complex when taking into account the agent's interaction with new agent environments such as the Semantic Web [2,3].

The Semantic Web extends the current web in such a way that the web pages' contents can be interpreted using ontologies [2] that help machines to understand web-content. Software agents can fulfil the interpretations in question by handling the semantic contents on behalf of their human users by collecting web contents from diverse sources, processing the information, and exchanging the results.

In addition, autonomous agents can evaluate semantic data and collaborate with semantically-defined entities of the Semantic Web, such as semantic web services (SWS), by using content languages [4]. SWSs can be simply defined as the web services with semantic interfaces to be discovered and executed [5]. In order to support the semantic interoperabilities and automatic compositions of web services, the capabilities of web services are defined in service ontologies that provide the required semantic interfaces. Such interfaces of the SWSs can be discovered by software agents and these agents may then interact with these services to complete their tasks. The engagements and invocations of a semantic web service are also performed according to the service's semantic protocol definitions. For instance, the dynamic composition of heterogeneous services for the optimal selection of service providers can be achieved by employing agents and ontologies, as proposed in [6].

However, agent interactions with semantic web services add further complexities when designing and implementing MASs. Different methodologies can be applied in order to deal with this complexity. One of the possible alternatives is domain-specific languages (DSLs) [7–10] that have notations and constructs tailored towards a particular application domain (e.g. MAS and Semantic Web). In this way, DSLs raise the abstraction level, expressiveness, and ease of use. In other words, when using this methodology the users focus on the program model of the solution instead of a platform code for the solution, and they can automatically generate code from the model using DSL tools. As a result, the DSLs' users mostly need the knowledge from the problem domain [11] but little programming experience.

We are convinced that the use of a DSL can provide the required abstraction and support a more fruitful methodology for the development of MASs especially when working within a Semantic Web environment. Within this context, prior to the work discussed here, we first provided

* Corresponding author at: 208, International Computer Institute, EGE University, Bornova, Izmir-Turkey. Tel.: +905419188836.
E-mail addresses: moharram.challenger@mail.ege.edu.tr, m.challenger@gmail.com (M. Challenger), marjan.mernik@um.si (M. Mernik), geylani.kardas@ege.edu.tr (G. Kardas), tomaz.kosar@um.si (T. Kosar).

http://dx.doi.org/10.1016/j.csi.2015.08.012
0920-5489/

a metamodel defined in several viewpoints [12] for MASs working within a Semantic Web environment. Then, based on this metamodel, we developed a DSL called the Sematic web-Enabled Agent Language (SEA_L) [13,14] including an interpreter mechanism for SEA_L which is defined for enabling code generation regarding the implementation of SEA_L agents (e.g. in JADEX platform [15]).

Although syntax definition based on a metamodel is an essential part of a modelling language, an additional and required part is the determination and implementation of DSL constraints that constitute those semantics that cannot be defined solely by a metamodel or syntax. In line with these constraints, the semantics of a DSL include some rules that restrict the instance models created according to the language.

The definition of the semantic rules was provided towards this end in our previous study [16,17] focusing on Agent–Semantic Web Service interaction. However, the represented semantic rules were not implemented and applied to SEA_L language (although, it is one of the crucial tasks during a DSL's development). In other words, the defined formal semantic rules have not been applied to any translational/operational semantics and have no effects on the quality of the generated code. Such an application is important as the execution of the language is described directly or by translating to another language in translational/operational semantics.

Therefore, in this study, the specification of the static semantics regarding the interactions between software agents and semantic web services in SEA_L language is formally declared using denotational semantics [18] which paves the way for the implementation of these specifications using attribute grammars [19,20]. Static semantics is used to control some constraints during system modelling in the DSL. On the other hand, denotational semantics interpret the phrases of a language as mathematical denotations and conceptual meanings that can be thought of abstractly.

Our main motivation is that declarative specifications can present the meanings of both associations and constraints for the language in a formal way. The formal representation of the semantics helps to provide an unambiguous definition and precise meaning of a program. It also helps to have the possibility for more accurate code generation by the language-based tools [21], when they are implemented in the DSL (which is lacking in the related work). A successful system verification and validation can also be achieved with a proper formal semantics definition.

In order to implement the defined declarative semantics of SEA_L, we have employed the LISA tool [22] which is based on attribute grammars. As can be noticed in further sections of this paper, implementation using LISA enables the separation of concerns between the syntax and the semantics. Moreover, it offers tools for syntax and semantics evaluation separately.

Hence, the main contributions of this paper are listed as follows:

- a new DSL-based methodology for the development of MAS with Semantic Web, where DSLs are formally specified with formalism that enables incremental specifications; and
- the implementation of the static semantics of SEA_L MAS development language and integrating them within the operational semantics of SEA_L in the form of declarative specifications using attribute grammars; this leads to productivity and advanced semantic controls in our DSL, and modularity and extendibility for the new language.

The remainder of the paper is organised as follows: Section 2 discusses the related work. The applied approach is elaborated in Section 3. The proposed methodology is discussed in Section 4 by especially taking into consideration the Agent–SWS interaction viewpoint of the system. In Section 5, use of the language and its semantics is presented using a case-study. Section 6 covers the evaluation and discussion for this study. Finally, the paper is concluded in Section 7.

2. Related work

The design and implementation of MAS working on the Semantic Web keep their emphasis since the first introduction of this new generation web in [2]. Berners-Lee et al. [2] took software agents as the central point of distributed content collection, knowledge formalisation, processing and interpretation of data, which are all required for the realisation of such a web environment. Hence MAS composed of many autonomous agents now became one of the major components of the Semantic Web. More specifically, the integration of agents and the knowledge ontologies steers the use of web services [23] and enables the automatic discovery and execution of services by the agents within the Semantic Web environment. Many researchers have investigated how the agents can participate in service execution inside the Semantic Web environment and have provided noteworthy methodologies and/or protocols for collaboration between agents and other Semantic Web entities, e.g. semantic web services. For instance, Paolucci et al. [24], Sycara et al. [5] and Li and Horrocks [25] proposed various capability representation mechanisms for semantic web services and discussed how they can be discovered and executed by agents. Agents infer about the suitability of the advertised semantic web services for the required action according to those defined service representations and decide on executing the more appropriate service. The studies in [26] and [27] described agent environments which use OWL-S ontologies to advertise descriptions of agent services to transport them using communication messages. Those descriptions provided for the use of agent services as if they were semantic web services. OWL-S [28] is an ontology built on top of Web Ontology Language (OWL) for describing Semantic Web Services and enables users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. In addition, OWL [29] is a family of knowledge representation languages for authoring ontologies which are a formal way of describing taxonomies and classification networks.

A set of architectural and protocol abstractions that serves as a foundation for Agent–web service interactions on the Semantic Web was introduced in [30]. This initiative architecture addressed the requirements of dynamic service discovery, service engagement, service process enactment and management, community support and quality of service for the Semantic Web. The architecture is based on the MAS infrastructure because the specified requirements can be accomplished with asynchronous interactions and using goal-oriented software agents. Two implementations of this conceptual architecture were provided in [31] and [32] which mainly considered the service matchmaking, service discovery and service execution functionalities of software agents. Preparation of existing web services into the Semantic Web environment or migrating them via software agents is also possible by following approaches such as given in [33] and [31].

Similar to [5] and [25], Talantikite et al. [34] provided an input-output similarity measure for OWL-S ontologies for determining the best semantic services and compose them for meeting client requests. Instead of semantic web service profiles, the use of OWL-S process models is proposed during the service discovery in [35]. Hence, it is aimed at finding and matching more relevant services with the proposed algorithm. Kumar [36] discussed multi-attribute negotiation between the agents working on the Semantic Web and the benefits of such negotiation on both the selecting and composing of semantic web services. Our study contributes to the above-mentioned MAS and the Semantic Web research by providing a DSL which can be used to formally define and implement the interaction between software agents and semantic web services. MAS developers can use the specifications given in this paper to realise concrete implementations of collaboration protocols and abstract Semantic Web service architectures (e.g. [30] and [36]) discussed above; such that agents may interpret and reason with semantic descriptions during the deployment of semantic web services.

On the other hand, Agent-oriented Software Engineering (AOSE) is one of the major areas of agent research intersecting Software