The Role of Ontology in Modelling Autonomous Agent-Based Systems

Ebrahim Alhashel, Bala M. Balachandran, and Dharmendra Sharma

School of Information Sciences and Engineering
University of Canberra, ACT 2601, Australia
{ebrahim.al.hashel,bala.balachandran,
dharmendra.sharma}@canberra.edu.au

Abstract. An agent-based system is characterised by an agent's autonomous behaviour, which behaviour is the main difference between the concepts of agent and object. Agent autonomous behaviour is the ability of an agent to cooperate instead of integrate; therefore, the structure of agent-based systems consists of loosely coupled agents. In such an environment, the relationship between the agents is unlocked, so conventional, predefined integration software techniques are not an option because the agents need an open-architecture type of integration (cooperation) to achieve their tasks jointly. The aim of this research paper is to provide an evidence of how the ontology approach can play a role in modelling agent autonomous behaviour. The research explores the ontology software technologies used for semantic web applications, and designs a case study as an example of a set of services. In the implementation phase, the research uses the web ontology software development languages XML, RDFS, OWL, and Altova semanticWork to set up and develop the case study. The result is presented and plans for future work are discussed.

Keywords: Software engineering, knowledge-based system, agent-oriented software development, AOSE, Ontology, semantic Web, Web Ontology Language.

1 Introduction

Agent-based computing offers a new software design for managing the inherent complexity of software systems [1]. Unlike an object-based system or any other software paradigm, an agent-based system is characterised by agent autonomous behaviour wherein the agent can act without predefined rigid manipulation or links, such as object invocation and primary and foreign key techniques. To design agent autonomous behaviours, there must be an open integration architecture that permits an agent to function in its environment and to cooperate with other agents. The theory of the autonomous agent specifies that the agents must have the ability to act in a group toward solving a common problem—in other words, to act cooperatively [2].

This paper presents the concepts of Ontologies and the existing ontology-based software development tools to increase understanding of how efficiently this technology can be utilised for developing agent-based paradigms, particularly in

J.D. Velásquez et al. (Eds.): KES 2009, Part II, LNAI 5712, pp. 111–118, 2009.

modelling agent autonomous behaviour. Given the view that agent behaviour requires an open-architecture mechanism, we consider the ontology techniques to be a powerful tool to represent agent autonomous concepts. The idea is that, *if the agent can realize the meaning of things (objects), can match its own capability with these objects' meanings, and then can identify its domain objective, this is an efficient technique to support the agent autonomy*.

To illustrate the research vision, we develop a travel agency system (TAS) e-commerce case study defined in [3], and then demonstrate the implementation processes. The architecture of TAS is divided to two main components: the services and the agents. The services are the subsystems, for example, Car, Hotel, and Flight. The services must be ontologically defined and then made readable and understandable for the system's agents. The agents must be able to discover the services' functionalities (what the services do exactly). The agents are designed in two categories: professional agents and skill agents. The professional agents are responsible for the main user goal (what the user intends the system to do), for example, travel to Melbourne. The skill agents are responsible for representing a group of services with the same functionalities, for example, a set of services providing cars for hire.

The ontology of the system's services provides an efficient search based on the functionality (what the service does), rather than only on the service name (key word). The interpretation of the inter-operations of the services is the main element required to create automatic configuration between the services because this interpretation process allows one service to use other services' facilities to satisfy the user's goal. For example, a holiday package needs integration among three agent services: hotel booking, flight reservation, and car hire.

This paper is organised as follows. Section 2 discusses the agent autonomous architectures. Section 3 briefly demonstrates the development of the ontology using the existing software development tools. Section 4 uses a TAS case study to illustrate how an agent-based system can be developed using ontology technologies. Section 5 summarises our conclusions and relates plans for future work.

2 Autonomous Agent Architectures

Many agent-based software architectures have been proposed, with each focusing on a specific agent approach or designed toward a specific applications domain. This multiplicity of approaches has resulted in many different architectures. However, agent autonomous behaviour has not yet been implemented or designed under any architecture.

The agent-based software is a new paradigm for which some important questions remain, including which components constitute agent-based systems, to what level the agent concept has been implemented, how these concepts are represented and how they fit together to form the required system. There is no definite answer to these questions, but the existing agent development tools are affected by this state of affairs. For example, agent software engineering development methodologies such as Gaia [4] view the system according to an organisational structure approach. MaSE [5] combines several pre-existing models into a single structure methodology and then

focuses on goal hierarchy and system roles. Prometheus [6] develops BDI independent agents. Tropos [7] emphasises agent components as actors, goals, and plans. ROADMAP [8] is designed from knowledge perspectives and is normally used as an extension of Gaia [3].

Other agent-based approaches can be examined in terms of their development environments. JADE [9], for example, uses FIPA agent architecture. JADEX [10] is a Belief Desire Intention (BDI) reasoning engine that allows for programming intelligent software agents in XML and Java. JACK [11] develops BDI agent architecture [12]. Cougaar [13] is used for designing and building Society-Community agent architecture, and Aglet [14] deals with the mobility of an agent.

There are also design-objective models, such as RETSINA [15], which generates a multi-agent architecture based on peer coordination. The Zeus [16] agent development tool uses a graphical user interface (GUI) to define agent components and generate java code. A few theoretical attempts have been made in the same stream. Luck [17] discussed an alternative state architecture and, by using adoption strategy, showed how agents emerge from objects. All of these attempts have endeavoured to formulate software agent architecture using different approaches. For more details on these approaches, refer to [18].

3 Ontology in Software Engineering

The use of ontology in software engineering comes into demand in active research areas. The new trend in software engineering is to construct a logic-ware that possesses a "reason" property in its processing and, in this, ontology plays a main role and can be used to represent a meaning (knowledge) of the application domain. It has been shown that, in merging the ontology into the software, engineering ontology using UML has some drawbacks. To show the use of UML, class diagrams will be used to represent concepts and their attributes and relationships between concepts. For axiom, an Object Constraint Language (OCL) is used and attached to the concept notes, since UML lacks formal semantics [19]. [20]. However, UML does not own the full potential to model the Ontologies domain model and, for this reason, this research paper introduces its own modelling diagram to represent both agent abstractions and the ontology constraints.

Other software engineering for modelling ontology can be done by extending the database design techniques and the entity-relationship diagram (ERD) [21]. Asuncion et al. demonstrated the utilisation of ERD, and then mentioned the use of the model extension using HERM (high-order entity-relationship model) to add complex attribute types (key constraints, generalisation and specialisation relationships, etc.) [22]. However, the main drawback of using extended ERD is that heavyweight ontology cannot be modelled. A set of ontology development tools is currently available. This research focuses only on the relevant tools, specifically those based on eXtensible Markup Language (XML), such as Ontology Web Language (OWL) [23] and its extension, Resources Description Framework (RDFS). OWL provides more advance inference to answer queries not necessary predefined properly since OWL has more

advanced generalisation properties than RDFS. OWL is part of the growing stack of W3C recommendations related to the Semantic Web. The following are more illustrations of the use of semantics, taken directly from [23]:-

- XML provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a datamodel for objects ("resources") and the relationships between them and provides a simple semantics for this datamodel, which can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalisation-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relationships between classes (e.g., disjointedness), cardinality (e.g., "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes.

4 Case Study

Planning a travel package that involves a flight, a hotel, and a car reservation on a given date and time is a complex task to automate, particularly when more than one website is involved. Coordination between each service is the key to accomplishing a task such as this. Currently, this type of task is performed by humans after searching and accessing several websites and making decisions based on preferences. In this environment, the concept of an intelligent software agent has great potential for helping the customer get the best deal on a travel package.

The proposed **Travel Agency System (TAS)** is a multi-agent system designed to obtain travel packages for users based on their preferences. This process begins when the TAS scenario is defined to capture the goals and sub-goals. For this purpose, the extended Prometheus development methodology defined by [24] is applied. For the result, see Figures 1 and 2.

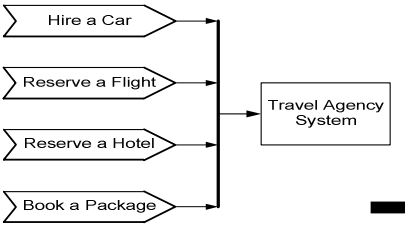

Fig. 1. System goals diagram

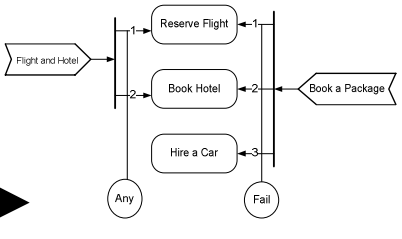

Fig. 2. Converting goals to tasks

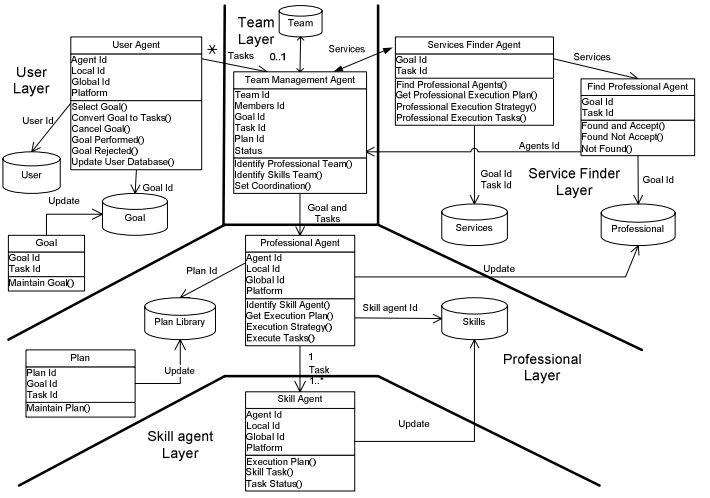

Fig. 3. TAS agent classes architecture

It is not practical to show all the implementation details here, but Figure 3 shows the TAS class diagram based on the TAS architecture. The TAS architecture is divided into five layers: User agent, Services finder, Team management, Professional agent, and Skill agent. The main design principle used in TAS architecture is that partitioning the services layer from the team management layer isolates services (what will be done) from the execution (who will do it). This partitioning is made possible because services are defined semantically using resources description framework language (RDF) in parallel with ontology web language (OWL) classes, and extensible mark-up language (XML) for data representations.

TAS User Layer: In this process, the user's goals are captured and then analysed further until all the goals and related tasks are identified.

Team Management Layer: At this layer, both the system agent and the agent components are identified, and the team database, the plan database, and the professional agents required to achieve the goals are recognised.

Services Finder Layer: The functionality model for each service should be developed with the execution sequence. (Refer to the elements of the service profile listed in [25].) Each row has to be defined at this stage, and the XML document that represents the services data that will be used by OWL has to be developed. Figure 4 provides a selected segment code for the car rental ontology.

Professional agent layer: According to the TAS scenario, the goal layer consists of three professional agents: Car, Flight, Hotel, and Package. The main elements of the agent inter-structure have to be developed for every agent (Figure 5).

Skill agent Layer: The Skill agent layer is concerned with three aspects of the process: the professional agent under which it performs, the execution sequence, and which services it executes. The plan and the database of the TAS are also considered at this stage.

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rent="http://www.rent.com/ontologies/Rent#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

 <rdf:Description
  rdf:about="http://www.rent.com/ontologies/Rent#Rent">
  <rdf:type>
   <rdf:Description
    rdf:about="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:type>
 </rdf:Description>
 <rdf:Description
  rdf:about="http://www.rent.com/ontologies/Rent#Car">
  <rdf:type>
   <rdf:Description
    rdf:about="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:type>
  <rdfs:subClassOf>
   <rdf:Description
  rdf:about="http://www.rent.com/ontologies/Rent#Rent"/>
   </rdfs:subClassOf>
 </rdf:Description>
 <rdf:Description
  rdf:about="http://www.rent.com/ontologies/Rent#Type">
  <rdf:type>
   <rdf:Description
    rdf:about="http://www.w3.org/2002/07/owl#Class"/>
   </rdf:type>
 </rdf:Description>
 <rdf:Description
  rdf:about="http://www.rent.com/ontologies/Rent#Brand">
  <rdf:type>
   <rdf:Description
    rdf:about="http://www.w3.org/2002/07/owl#Class"/>
   </rdf:type>
 </rdf:Description>
 .
 .
  </rdf:RDF>
```

Fig. 4. Selected segment of OWL codes for car rent ontology

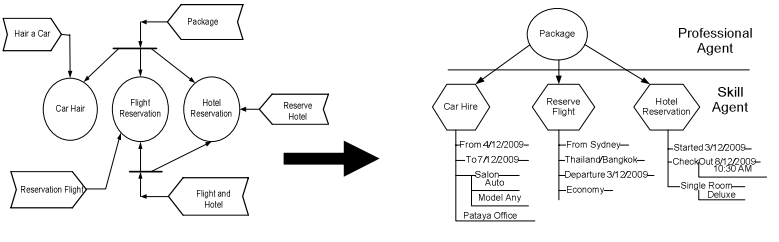

Fig. 5. Relationships between professional agent, Skill agent, and goals

5 Conclusion and Future Work

The agent-based system introduces a new feature that adds challenges to software engineering. Many tools and methodologies have been developed to engineer agent behaviour, and all have their advantages and disadvantages. Developing an ideal agent-based system agent's autonomous behaviour requires agents to cooperate instead of predefining integration; otherwise, agent becomes object.

The TAS architecture is an agent-based system designed to enable agent to exercise its autonomous behaviour. To implement agent autonomous feature, partitioning the system's characteristics from the agent's actions is essential; in TAS, the system characteristics are partitioned from the agent's acts by using a services-related semantic approach and the web ontology tools OWL, RDFS, and XML. This technique provides an efficient methodology for developing cooperative software agent-based systems. TAS is currently in the implementation phase, so the result of this project will be reported in the near future.

References

[1] Odell, J.: Objects and Agents Compared. Journal of Object Technology 1, 41–54 (2002)
[2] Alhashel, E., Mohammadain, M.: Illustration of Multi-agent Systems. Presented at CIMCA, Austria - Vienna (2008)
[3] Alhashel, E., Balachandran, B., Sharma, D.: Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part III. LNCS (LNAI), vol. 4694, pp. 909–916. Springer, Heidelberg (2007)
[4] Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Presented at Autonomous Agents and Multi-Agent Systems, The Netherlands (2000)
[5] Wood, M.F., Deloach, S.A.: An Overview of the Multi-agent System Engineering Methodology. Presented at First International Workshop on Agent-Oriented Software Engineering, Limerick Ireland (2001)
[6] Padgham, L., Michael, W.: Developing Intelligent Agent Systems, vol. 1, pp. 23–82. John Wiley & Sons, Ltd, England (2004)
[7] Bresciani, P., Giorgini, P., Giunchiglia, F., et al.: Tropos: An Agent-Oriented Software Development Methodology. Presented at Autonomous Agents and Multi-Agent Systems, Netherlands (2004)
[8] Juan, T.: The ROADMAP Meta-Model for Intelligent Adaptive Multi-Agent Systems in Open Environments (2003)
[9] Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-agent Systems with JADE. Wiley, Chichester (2007)
[10] Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A BDI-Agent System Combining Middleware and Reasoning. In: Software Agent-Based Applications, Platforms and Development Kits. Whitestein Series in Software Agent Technologies and Autonomic Computing, pp. 143–168. Birkhäuser, Basel (2005)
[11] JACK Development Toolkit (07/07/2008), http://www.agent-software.com/

[12] Kinny, D., Georgef, M., Rao, A.: A Methodology and Modelling Technique for System of BDI Agents. Presented at Modelling Autonomous Agents in a Multi-Agent World MAAMAW 1996, Germany (1996)
[13] Cognitive Agent Architecture (Cougaar) Project (17/04/2009), http://www.cougaar.org/
[14] Aglet Development Toolkit (23/07/2008), http://aglets.sourceforge.net/
[15] RETSINA Development Toolkit (24/06/2008), http://www.ri.cmu.edu/projects/project_76.html
[16] Development toolkit (12/09/2008), http://labs.bt.com/projects/agents/zeus/
[17] Luck, M., d'Inverno, M.: A Formal Framework for Agency and Autonomy. Presented at International Conference on Multi-Agent Systems (1996)
[18] Luck, M., Ashri, R., D'Inverno, M.: Agent-Based Software Development. Artec House Inc., London (2004)
[19] Siricharoen, V.W.: Ontologies and Object models in Object Oriented Software Engineering. Presented at International Multi Conference of Engineers and Computer Scientists 2007, Hong Kong, March 21-23 (2007)
[20] Gomez-Perez, A., Fernandez-Lopez, M., Corcho, O.: Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. Springer, Heidelberg (2004)
[21] Fankam, C., Jean, S., Bellatreche, L., et al.: Extending the ANSI/SPARC architecture database with explicit data semantics: An ontology-based approach. In: Morrison, R., Balasubramaniam, D., Falkner, K. (eds.) ECSA 2008. LNCS, vol. 5292, pp. 318–321. Springer, Heidelberg (2008)
[22] Thalheim, B.: Entity-Relationship Modeling: Foundations of Database Technology. Springer, Berlin (2000)
[23] OWL Web Ontology Language (01/06/2009), http://www.w3.org/TR/owl-features/
[24] Alhashel, E., Balachandren, B., Sharma, D.: Enhancing Prometheus to Incorporate Agent Cooperation Process. Presented at International Conference on Computational Intelligence for Modelling, Control and Automation - CIMCA, Austria - Vienna (2008)
[25] OWL-S: Sementic Markup for Web Services (10/04/2009), http://www.w3.org/Submission/OWL-S/