Formal Specification Language for Agent Oriented Systems

I

Vinitha Hannah Subburaj
Baldwin Wallace University, USA

Joseph E. Urban
Texas Tech University, USA

INTRODUCTION

Software engineering is a developing discipline in engineering. Traditional software engineering methods focus on development of reliable software going through sequential software life cycle phases. Formal methods in software requirements specification use mathematical rules and a fixed set of syntax and semantics to specify software systems. The validation of software requirements specification after implementation is also made possible through the application of formal methods to software requirements specification. Agent oriented software engineering involves the use of agents to realize an entire system. Agents are autonomous and are used to accomplish a specific task. The characteristics and behavioral properties of agent systems involve a high level of complex design. The design and development of such complex systems can become more challenging by the use of traditional software engineering practices. Using formal methods to specify the agent systems can result in highly reliable systems.

A few of the concerns addressed for the use of formal methodologies along with software requirements specifications are as follows: ease of use and understandability rendered by the formal approach; level of expertise required to use the formal approach to specify software systems; and existence/development of tool support to apply formal methodologies to software specifications and to validate the specification after development using tool support. These concerns do not address the complete set, but address a few of the important factors. This article draws the attention of the readers to the use of formal methods in the software requirements specification phase by addressing those concerns. Agent oriented software systems is a software engineering domain that is gaining importance due to the vast growth of information around the world. The development of agent oriented software systems seems to be inclined towards the application of formal methodologies due to the level of preciseness and abstraction required to address detailed functionalities of the systems. Formal methodologies applied while specifying agent oriented software systems can result in the development of correct and reliable agent systems.

This article gives an introduction to formal specification languages and discusses ways of using formal methods to specify software agent systems. Agent oriented methods are becoming popular and are applied to a wide variety of application domains. Due to this diversity, the notion of agents along with their behavioral characteristics will be discussed in the beginning to define the scope and boundaries. The agent architecture and the characteristics are discussed while developing the formal specifications. The article has been organized into two broad categories: addressing the concerns in the application of formal methods during the software specification phase of software development along with ways to overcome them; and application of formal methodologies to specify complex agent systems.

BACKGROUND

This section describes related work that exists in applying formal methods to specify software systems in specific agent oriented software systems. The state of the art with respect to the agent oriented software specification is also described in this section.

de Vries et al. (de Vries et al., 2001) introduced a new operational model of agents that describes the

DOI: 10.4018/978-1-4666-5888-2.ch404

interaction of agents with each other and with the environment. An agent is viewed as a program that has a state. An agent's state has two parts: a sense buffer that is used to store the observed and communicated information, and the mental state that consists of a set of mental formulas. Kinny (Kinny, 2001) introduced an algebraic approach that describes the operational semantics of agent computation. Kinny defined agents as a strong encapsulation of beliefs, intentions, and plans. The agent interacts with the environment using two interfaces namely: sensors and effectors. The framework defined by Kinny follows an algebraic approach. The computation specified in the algebraic approach is kept separate from the agent's program. Bosse et al. (Bosse et al., 2006) proposed the predicate logic Temporal Trace Language (TTL) for formally specifying multi-agent systems. Both quantitative and qualitative aspects of agent systems were specified using the Temporal Trace Language.

Rahman et al. (Rahman et al., 2008) introduced a formal model for an agent based supply chain framework in a virtual enterprise. Based on the requirements specification, a decision tree was used to capture the logic of production planning. A decision tree contains leaf nodes and non-leaf nodes. The leaves of the decision tree are labeled with classifications and the non-leaf nodes are labeled with attributes. Weiss et al. (Weiss et al., 2006) introduced a formal language, Autonomy Specification Language (ASL), for formally specifying the agent activities along with the set of constraints on these activities. The syntax of ASL has been described using extended Backus-Naur form. A multi-agent system was developed by Bagic (Bagic, 2004) with the application of a methodology that extends AUML with Petri nets. The work focused on analyzing the connecting points between current AUML specifications with concepts of Petri nets. AUML was used as a main modeling tool for a multi-agent system and Petri nets as a formal verification and validation tool. Luck and d'Inverno (Luck & d'Inverno, 1995, 2001) introduced a formal and conceptual framework for defining agent systems. The framework for defining agents was introduced by the application of formal methods. The Z specification language was used to formally specify agent systems. A high level of abstraction and incrementally increasing the specification details of agent systems was presented by the framework. The Z language allowed for the use of mathematical notations to formally specify agent systems.

Guan and Ghose (Guan & Ghose, 2005) proposed an executable specification framework for agent oriented conceptual modeling. The two kinds of graphical models used in the i* framework are: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model captures the social context and the SR model captures the internal characteristics of the actors in a system.

Notations are symbols used in a formal methodology to represent elements in a system. A modeling technique describes a set of models used in the methodology to depict a system with different levels of abstraction. The list of properties used to assess the notations and modeling techniques as given by Sturm and Shehory (Sturm & Shehory, 2004) are accessibility, analyzability, complexity management, executability, expressiveness, modularity, and preciseness. A comparative analysis of the above mentioned existing agent methodologies based on notation and modeling properties as specified by Sturm and Shehory indicates that not all formal agent specification languages satisfy the executability criterion and the complexity management criterion. The complexity management criterion deals with the ability of a methodology to deal with different levels of abstraction. The extended Descartes specification language uses a simple tree structuring method with indentation making it easy to read and understand. The extended Descartes specifications language specifications are written in a top-down modular approach. The case study presented in this article for specifying agent systems can be used to determine the ease of understanding of the specifications written using the extended Descartes specification language. The agent framework described in this article addressed both low level and high level abstraction concepts of an agent system satisfying the complexity management criterion. The extended Descartes specifications are executable using the implemented interpreter.

The Descartes specification language (Urban, 1977) was designed for use throughout the software life cycle. The relationship between the input and the output of a system is functionally specified when using this specification language. Descartes defines the input data and output data and then relates them in such a way that output data becomes a function of input data. The data structuring methods used with this language are known as Hoare trees. These Hoare trees use three structuring methods, namely direct product, discriminated union, and sequence.