Formal Development of Reactive Agent-Based Systems

P. Kefalas
CITY College, Greece

M. Holcombe
University of Sheffield, UK

G. Eleftherakis
CITY College, Greece

M. Gheorghe
University of Sheffield, UK

INTRODUCTION

Recent advances in both the testing and verification of software based on formal specifications have reached a point where the ideas can be applied in a powerful way in the design of agent-based systems. The software engineering research has highlighted a number of important issues: the importance of the type of modelling technique used; the careful design of the model to enable powerful testing techniques to be used; the automated verification of the behavioural properties of the system; and the need to provide a mechanism for translating the formal models into executable software in a simple and transparent way.

An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives (Jennings, 2000). There are two fundamental concepts associated with any dynamic or reactive system (Holcombe & Ipate, 1998): the environment, which could be precisely or ill-specified or even completely unknown and the agent that will be responding to environmental changes by changing its basic parameters and possibly affecting the environment as well. Agents, as highly dynamic systems, are concerned with three essential factors: a set of appropriate environmental stimuli or inputs, a set of internal states of the agent, and a rule that relates the two above and determines what the agent state will change to if a particular input arrives while the agent is in a particular state.

One of the challenges that emerges in intelligent agent engineering is to develop agent models and agent implementations that are "correct." The criteria for "correctness" are (Ipate & Holcombe, 1998): the initial agent model should match the requirements, the agent model should satisfy any necessary properties in order to meet its design objectives, and the implementation should pass all tests constructed using a complete functional test-generation method. All the above criteria are closely related to stages of agent system development, i.e., modelling, validation, verification, and testing.

BACKGROUND: FORMAL METHODS AND AGENT-BASED SYSTEMS

Although agent-oriented software engineering aims to manage the inherent complexity of software systems (Wooldridge & Ciancarini, 2001; Jennings, 2001), there is still no evidence to suggest that any methodology proposed leads toward "correct" systems. In the last few decades, there has been strong debate on whether formal methods can achieve this goal. Software system specification has centred on the use of models of data types, either functional or relational models, such as Z (Spivey, 1989) or VDM (Jones, 1990), or axiomatic ones, such as OBJ (Futatsugi et al., 1985). Although these have led to some considerable advances in software design, they lack the ability to express the dynamics of the system. Also, transforming an implicit formal description into an effective working system is not straightforward. Other formal methods, such as finite state machines (Wulf et al., 1981) or Petri Nets (Reisig, 1985) capture the essential feature, which is "change," but fail to describe the system completely, because there is little or no reference to the internal data and how these data are affected by each operation in the state transition diagram. Other methods, like statecharts (Harel 1987), capture the requirements of dynamic behaviour and modelling of data but are informal with respect to clarity and semantics. So far, little attention has been paid in formal methods that could facilitate all crucial stages of "correct" system development, modelling, verification, and testing.

In agent-oriented engineering, there have been several attempts to use formal methods, each one focusing on different aspects of agent systems development. One was to

formalise the PRS (procedural reasoning system), a variant of the BDI architecture (Rao & Georgeff, 1995), with the use of Z, in order to understand the architecture in a better way, to be able to move to the implementation through refinement of the specification, and to be able to develop proof theories for the architecture (D'Inverno et al., 1998). Trying to capture the dynamics of an agent system, Rosenschein and Kaebling (1995) viewed an agent as a situated automaton that generates a mapping from inputs to outputs, mediated by its internal state. Brazier et al. (1995) developed the DESIRE framework, which focuses on the specification of the dynamics of the reasoning and acting behaviour of multiagent systems. In an attempt to verify whether properties of agent models are true, work has been done on model checking of multiagent systems with reuse of existing technology and tools (Benerecetti et al., 1999, Rao & Georgeff, 1993). Toward implementation of agent systems, Attoui and Hasbani (1997) focused on program generation of reactive systems through a formal transformation process. A wider approach is taken by Fisher and Wooldridge (1997), who utilised Concurrent METATEM in order to formally specify multiagent systems and then directly execute the specification while verifying important temporal properties of the system. Finally, in a less formal approach, extensions to Unified Modelling Language (UML) to accommodate the distinctive requirements of agents (AUML) were proposed (Odell et al., 2000).

X-MACHINES FOR AGENT-BASED SYSTEM DEVELOPMENT

An X-machine is a general computational machine (Eilenberg, 1974) that resembles a finite state machine but with two significant differences: there is memory attached to the machine, and the transitions are labeled with functions that operate on inputs and memory values. The X-machine formal method forms the basis for a specification language with great potential value to software engineers, because they can facilitate modelling of agents that demand remembering as well as reactiveness. Figure 1 shows the model of an ant-like agent that searches for food but also remembers food positions in order to set up its next goals. Many other biological processes seem to behave like agents, as, for example, a colony of foraging bees, tissue cells, etc. (Kefalas et al., 2003a; Gheorghe et al., 2001; Kefalas et al., 2003b). Formally, the definition of the X-machine requires the complete description of a set of inputs, outputs, and states; a memory tuple with typed elements; a set of functions and transitions; and finally, an initial state and a memory value (Holcombe, 1988).

Having constructed a model of an agent as an X-machine, it is possible to apply existing model-checking techniques to verify its properties. *CTL** is extended with memory quantifier operators: M_x (for all memory instances) and m_x (there exist memory instances) (Eleftherakis & Kefalas, 2001). For example, in the ant-like agent, model checking can verify whether food will eventually be dropped in the nest by the formula: $AG[\neg M_x(m_1 \neq \text{none}) \vee EFM_x(m_1 = \text{none})]$, where m_1 indicates the first element of the memory tuple.

Having ensured that the model is "correct," we need to also ensure that the implementation is "correct," this time with respect to the model. Holcombe and Ipate (1998) presented a testing method that under certain design-for-test conditions can provide a complete test-case set for the implementation. The testing process can be performed automatically by checking whether the output sequences produced by the implementation are identical to the ones expected from the agent model through this test-case set.

Figure 1. An X-machine that models an ant.