Science of Computer Programming 78 (2013) 725–746

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

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

An application-level technique based on recursive hierarchical state machines for agent execution state capture

Giancarlo Fortino *, Francesco Rango

DEIS – University of Calabria, Via P. Bucci cubo 41c, 87036 Rende (CS), Italy

ARTICLE INFO

Article history:
Received 15 March 2011
Received in revised form 1 October 2011
Accepted 3 October 2011
Available online 23 October 2011

Keywords:
Software agents
Distilled StateCharts
Execution state capture
JADE
Strong mobility

ABSTRACT

The capture of the execution state of agents in agent-based and multi-agent systems is a system feature needed to enable agent checkpointing, persistency and strong mobility that are basic mechanisms supporting more complex, distributed policies and algorithms for fault tolerance, load balancing, and transparent migration. Unfortunately, the majority of the currently available platforms for agents, particularly those based on the standard Java Virtual Machine, do not provide this important feature at the system-level. Several system-level and application-level approaches have been to date proposed for agent state execution capture. Although system-level approaches are effective, they modify the underlying virtual machine so endangering compatibility. Conversely, application-level approaches do not modify any system layer but they provide sophisticated agent programming models and/or agent converters that only allow a coarse-grain capture of agent state execution.

In this paper, we propose an application-level technique that allows for a programmable -grain capture of the execution state of agents ranging from a per-instruction to a statement-driven state capture. The technique is based on the Distilled StateCharts Star (DSC*) formalism that offers an agent-oriented type of recursive hierarchical state machines. According to the proposed technique a single-threaded agent program can be translated into a DSC* machine, containing agent data, code and execution state, by preserving the original agent program semantics. The proposed technique can notably be applied to any agent program written through an imperative-style, procedural or object-oriented programming language. Currently, it is implemented in Java and fully integrated into the JADE framework, being JADE one of the most diffused agent platforms. In particular, agents, which are specified through a generic Java-like agent language, can be automatically translated into JADE agents according to the JADE DSCStarBehaviour framework by means of a translator tool. A simple yet effective running example is used to exemplify the proposed technique from modeling to implementation.

1. Introduction

Agent-based and multi-agent systems are developed around the concept of agent, a goal-directed, computational and interacting entity that acts on behalf of another entity (or entities) [40]. Such agent systems are supported by agent platforms that basically provide agent programming libraries and system-level services to support agent execution. An important system-level feature that an agent platform needs to include for the development of robust and fault-tolerant agent systems is agent state execution capture. In fact, agent checkpointing, persistency and strong mobility are mechanisms fully enabled

* Corresponding author.
E-mail addresses: g.fortino@unical.it (G. Fortino), frango@si.deis.unical.it (F. Rango).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.10.001

by agent state execution capture. Agent checkpointing [24] is a technique for inserting fault tolerance into agent systems. In particular, it basically consists of storing a snapshot of the agent execution state and, later on, using it for restarting the agent execution in case of its failure. Agent persistency is a mechanism allowing saving an agent along with its state into the mass memory as a file. Archived agents can be later on loaded and resumed. Strong mobility [18] is an important feature of an agent that enables the migration of the agent state, data and code. Such mechanisms constitute the basis for supporting more complex, system-wide agent policies and algorithms for fault tolerance, load balancing, and transparent migration [25].

Unfortunately, even after more than one decade of research on agent systems, the majority of the currently available agent platforms, particularly those based on the standard Java Virtual Machine (JVM) do not provide this enabling core feature at system-level. In fact, currently the standard JVM does not yet provide mechanisms to capture the state execution of Java processes. Nevertheless, several solutions have been to date proposed to overcome such an issue by providing transparent and non-transparent approaches. Transparent approaches allow for (semi)automatic capture of the agent execution state without explicit state "save and restore" carried out through manually inserted programming code as in the case of non-transparent approaches. So the transparent approaches are more effective as they allow focusing on the definition of the agent behavior without taking care of the points in which the agent state is to be captured as requested by the non-transparent approaches. The non-transparent approaches, in fact, cannot be used without exactly defining the capture points so that they are applicable only in the case of agent-driven capture and not for system-driven capture based on an agent run-time system able to capture the agent state in any points of its execution. Moreover the agent execution state of complex agents is sometimes not easy to be captured through non-transparent approaches and sometimes impossible without semantics changes of the agent behavior. This would complicate agent design and slow down prototyping of agent systems. The transparent approaches can be classified into system-level [6,7,28,33], converters [5,30,29,19,22] and model-based [41,17,39]. The system-level approach modifies the underlying virtual machine to capture the execution state at process/thread level; however, the modified virtual machine is usually not compatible for agents previously developed. Conversely, converters and model-based approaches are application-level approaches that rely on specific, sometimes sophisticated, agent programming models and/or on converters at bytecode-level or source-code-level to allow for agent execution state capture. Although they do not require any virtual machine modification, the grain of capture of the agent execution state is usually coarse so that capture can be carried out only at specific points of agent execution. In fact, capture is either automatically driven by specific statements (in agent converters) or programmed by exploiting the reference agent programming models. Both techniques can be used only for an effective agent-driven capture and not for an effective system-driven capture.

In this paper, an application-level technique for the agent-driven and the system-driven capture of agent state execution is proposed. Agent-driven capture can be based on key statements (checkpoint, persistence, move, clone), whereas system-driven capture is fine-grain and carried out on a per-instruction basis. A distinctive feature of the defined technique is the capture of the execution state in recursive and mutually recursive methods that can also contain key statements. The proposed technique is based on the Distilled StateCharts Star (DSC*) formalism that provides an agent-oriented type of recursive hierarchical state machine through which the agent program and its execution state can be represented. According to the technique, a single-threaded agent program formalized as a main method and ancillary methods is translated into a DSC* object, preserving the original agent program semantics. The technique is currently implemented in Java and integrated into the JADE framework [4]. In particular, an agent program, which is formalized through a simple Java-based Agent Language (JAL), is converted into a JADE agent compliant to a newly defined JADE behavior named DSCStarBehaviour. Conversion can be carried out in two modes: (i) driven by the statements *checkpoint, persistence, move* and *clone*; (ii) instruction by instruction. The so obtained JADE behavior not only has the same semantics as the original agent program but also incorporates its execution state at run-time. A simple yet effective example, concerning an agent-based version of the Fibonacci program, is used to exemplify the proposed technique.

The rest of this paper is organized as follows. In Section 2, related work is described according to an ad-hoc defined reference taxonomy. In Section 3, the DSC* formalism is defined as an enhancement of the Distilled StateCharts formalism [17,15]. Section 4 details the proposed technique that is then exemplified in Section 5 by means of a simple yet effective running example: the mobile Fibonacci agent. Section 6 describes the JADE-based DSC* agent framework which allows for the implementation of the proposed technique in Java to automatically produce agents executable on the JADE platform. Finally, conclusions are drawn and future work anticipated.

2. Related work

As introduced in the previous section, several approaches addressing the problem of the capture of the agent execution state for the purpose of agent migration, cloning, persistency and checkpointing have been to date defined. The aim of this section is to overview and compare such approaches through a simple yet effective reference taxonomy (see Table 1).

The available approaches can be roughly classified into transparent or non-transparent. Transparent approaches support the agent state execution capture through methods that allows for implicit or explicit identification of the points in which the agent state is to be captured and the implicit capture of the execution state at such points without requiring programmers to manually insert code for saving and restoring the execution state. Conversely, non-transparent approaches are based on an explicit identification of the points in which the agent state is to be captured and on manual "save and restore"