Engineering Applications of Artificial Intelligence 26 (2013) 2093–2104

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Robust design of multi-agent system interactions: A testing approach based on pattern matching

Celia Gutiérrez [a,*], Iván García-Magariño [b], Emilio Serrano [c], Juan A. Botía [d]

[a] Department of Software Engineering and Artificial Intelligence, Facultad de Informática, Universidad Complutense de Madrid, Madrid 28040, Spain
[b] Departamento de Ingeniería Informática y Organización Industrial, Facultad de Enseñanzas Técnicas, Universidad a Distancia de Madrid, Collado Villalba 28400, Spain
[c] Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Madrid 28040, Spain
[d] Departamento de Ingenieria de la Información y las Comunicaciones, Facultad de Informática, Universidad de Murcia, Murcia 30100, Spain

ARTICLE INFO

Article history:
Received 11 October 2012
Received in revised form
17 April 2013
Accepted 12 June 2013
Available online 15 August 2013

Keywords:
Agent-oriented software engineering
Communication
Debugging
Interaction
Multi-agent system
Testing

ABSTRACT

The definition of protocols between agents is not enough for guaranteeing the absence of undesirable communication in organizations and the presence of desirable ones in large multi-agent systems (MASs). This is a consequence of the complex system nature of MASs, which cause sophisticated behaviors to arise out of a multiplicity of relatively simple interactions among the independent agents composing them. With this motivation, this paper presents an approach for testing communication in MAS architectures. In this approach, designers are not only recommended to specify the desired communication protocols, but also the undesired patterns and organization structures in the agents' communications, allowing designers to define robust communication structures. For this purpose, this work presents (1) a language to define such patterns; (2) a set of already defined desired and undesired patterns which usually appear in general MASs; (3) a tool that allows developers to automatically detect these patterns in logs of MAS executions; and (4) a guideline that takes developers through the testing of the communications in MASs. The current approach is experienced with a case study, and the results show that the application of the current approach and the suppression of detected undesired patterns improve the effectiveness and efficiency of the corresponding MAS.

1. Introduction

A MAS is a system composed of multiple intelligent interacting agents, where an agent is a computer system which is able to perform independent actions in order to satisfy its design objectives (Wooldridge, 2002). MASs may be used to implement a wide range of computer systems such as the Grid, the Semantic Web, peer-to-peer systems, pervasive computing, and ambient intelligence (Serrano and Botia, in press). Agents in a MAS interact among themselves, typically by messages through a computer network, and make decisions based on these interactions. The testing of these interactions, being testing "the process of executing a program with the intent of finding errors" (Myers et al., 2004), is an extremely complex task. This complexity is mainly given by a well-known feature of agent technologies: their ability to generate unpredictable, complex and emergent behaviors from very simple rules (Bauer et al., 2009). This paper is focused on defining an approach to conduct testing activities within the process of MAS software development, and more specifically, in the design of the interactions among agents.

Let us consider a MAS with a set of n agents A. Let us also consider that each agent communicates with n_a agents in the system and finally that each agent sends m_a messages each time it communicates with another agent (including responses to received messages). Then, the number of elements of the set of messages M is the result of the following expression: $|M| = n \cdot n_a \cdot m_a$. If we consider as an example the very simple case in which agents send a single message to another agent for a consultation and they respond with a single message, then $n_a = (n-1)$ and the number of messages sent by each agent is $m_a=2$, values corresponding to a MAS with a very poor communication. In this case, $|M| = n \cdot (n-1) \cdot 2 = 2n^2 - 2n$. This example (Serrano et al., 2010) shows that the number of messages grows rapidly with the number of agents, even in systems with very low exchange of messages. Therefore, executions of concurrent systems become a complex task when the number of involved participants grows (Wu et al., 2010). In consequence, the principal line of investigation that has been followed in the testing and debugging of MASs is the automation of the testing of agent's interactions.

One of the most popular approaches to perform this automatic testing of interactions is the use of *Petri-nets* (Poutakidis et al., 2002). As the related works section explains, other proposals are

* Corresponding author.
E-mail address: cegutier@fdi.ucm.es (C. Gutiérrez).

0952-1976/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.engappai.2013.06.006

based on the use of the *Agent Unified Modeling Language* (AUML) (Odell et al., 2000), *Propositional dynamic logic* (Paurobally, 2003), *Statecharts* (Harel and Politi, 1998), or *Dooley graphs* (Parunak, 1996). These approaches employ formalisms to define the protocols which specify the interaction between agents and, afterwards, these definitions provide the developer with an automatic testing of the interactions. These are significant contributions to the Agent Oriented Software Engineering (AOSE), but they present a series of downsides that this paper attempts to deal with. These approaches merge the definition of a protocol with its testing. This is not practical when the possible faults in the interactions do not depend on the protocol but on the semantics exchanged or the specific execution of the MAS. For instance, a particular execution of the Foundation for Intelligent Physical Agents (FIPA) Iterated Contract Net Interaction Protocol (FIPA, 2001) may involve more than a hundred iterations. Is that amount of iterations a bug? this depends on the specific execution and the specific system. In addition, most approaches require the use of a concrete modeling language of the protocols and this reduces their usability, since developers are forced to use these if they want to address the testing of the interactions. Moreover, separating the definition of the protocols from the testing code, like the approach presented in this paper, the latter can be reused for several cases of the former, since the same bugs looked for in a protocol can be found in systems interacting by other protocols. Another added advantage of the separation between protocol definitions and testing code is that different developers can write them. This is a great advantage from the point of view of software engineering since it is often hard that developers detect all the mistakes made by themselves. Finally, desirable and undesirable recurring bugs in the design of MASs should be provided. These *patterns* would serve as "convenient design practices" to address the construction of these systems regardless of the specific modeling language employed.

This paper proposes an approach that assists designers in defining sets of tests to provide their MASs with robust communications. Desirable and undesirable patterns commonly found in the execution of these systems are provided in a formal and intuitive way. These patterns can be specific of a MAS (e.g. two types of agents that should not directly exchange information) or more general (e.g. deadlocks and redundancies). More specifically, the patterns presented are: (1) *deadlock*, to detect deadlocks between agents; (2) *inanition*, occurring when an agent keeps waiting a message from other; (3) *redundancy*, to deal with redundant interactions; (4) *Bias Agent*, bias for an agent to communicate with the same agent; (5) *Bias Type*, bias for agents of a specific type to communicate with the same agent; and (6) *unexpected*, unexpected sequence of agent types regarding it as the sequence of message senders in the same conversation. Due to the versatility of MASs, these patterns are not fixed. Therefore, an essential part of the testing approach presented in this paper, which takes the developer through the testing of the interactions in a specific execution of a concrete system with a particular semantics, is the adaptation of the general patterns to the specific case.

The main components of the testing approach presented in this paper are: (1) a formal language for specifying communications patterns; (2) a corpus of general behaviors defined with this language that are usually undesirable; (3) a tool to detect these patterns automatically from logs of MAS executions; and (4) a guideline to apply the approach and achieve robust communications in MASs. We have a previous work in Gutiérrez and García-Magariño (2011) with the experimentation to a case study. With the presented work, we provide a robust approach, with its application to a different case study. Our aim is to illustrate the usefulness of this proposal and its suitability for MASs built by different agent-oriented methodologies. Furthermore, the new presented case study has been measured before and after applying the presented approach and suppressing the detected undesired patterns, and the results show that the current approach improved the effectiveness and performance of the communications of the MAS of this case study. Unlike other previous works (Serrano et al., 2013, the approach presented is syntax-oriented since the usability and coverage are our main concern. The proposal can be easily employed in most conceivable MASs without design constraints to deal with the interactions information. For that purpose, the approach is based on simple fields which are always present in these systems or trivially added. Concretely, the messages contain: timestamp, sender, receivers and a specific performative.

In brief, the structure of this paper is the following: Section 2 covers the related works in this area; Section 3 introduces the presented approach (the formal language, the patterns in interactions based on this language, the tool for detecting the patterns, and the guideline for testing communications); Section 4 describes the experimental results based on the development of a MAS and measures its improvements of effectiveness and efficiency; and finally, Section 5 mentions the conclusions and future work.

2. Related works

The main research line that has influenced the present work is the automatic testing of agent's interactions (Serrano et al., 2010). This kind of approaches usually needs three steps: (1) to define the protocols which specify the interaction between agents; (2) to automatically test whether these protocols were correctly performed, based on whether the agents do not violate the specifications; and (3) to locate the errors found through some sort of display.

Petri-nets are one of the most popular methods to specify and debug communications between agents (Mazouzi et al., 2002; Cost et al., 2000; Nowostawski et al., 2001). They consist of a generalization of automata theory so as to be able to express events occurring simultaneously. Poutakidis et al. (2002) propose specifying protocols with AUML and translating them into Petri-net formalism. In the same vein, Miller et al. (2010) defines two test coverage criteria for agent interaction testing: protocol-based coverage criteria and plan-based coverage criteria. The former use a protocol specification and the latter also needs plans that interact with the protocol. Again, the protocols are defined by using Petri Nets which, besides, are annotated to measure the mentioned test coverage criteria.

AUML (Odell et al., 2000) is an extension of the UML notation for specifying interaction protocols for agents. AUML provides a range of features such as merging of messages, a non-exclusive choice, and the ability to specify sequencing of messages. AUML and Petri-nets are utilized by the debugger in the contribution mentioned to monitor conversations and provide error messages when protocols are not followed correctly. With this approach, the same authors classify typical mistakes that can be found in MASs (Poutakidis et al., 2003): uninitialized agent, failure to send, wrong recipient, message sent multiple times, and wrong message sent.

Besides the AUML diagrams and Petri nets, other approaches for the automatic testing of protocols use:

- *Propositional dynamic logic* (Paurobally, 2003). Dynamic logic is an extension of modal logic. The extension proposed by the authors is called the Agent Negotiation Meta-Language (ANML). Interaction protocols in ANML are in the form of multi-modal theories, leading to an abstract theory of an interaction in a group and allowing the automatic testing of these protocols.
- *Statecharts* (Harel and Politi, 1998). A statecharts diagram is a type of diagram used to describe the behavior of systems.