Available online at www.sciencedirect.com
SCIENCE DIRECT®
Knowledge-Based Systems 19 (2006) 235–247
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

# A new approach in development of distributed framework for automated software testing using agents

P. Dhavachelvan [a,*], G.V. Uma [b], V.S.K. Venkatachalapathy [c]

[a] Department of Information Technology, Thiruvalluvar College of Engineering and Technology, Tamilnadu, India
[b] Department of Computer Science and Engineering, Anna University, Tamilnadu, India
[c] Department of Mechanical Engineering, Sri Manakula Vinayagar Engineering College, Pondicherry, India

Received 14 November 2003; accepted 16 December 2005
Available online 10 February 2006

**Abstract**

Software testing is the technical kernel of software quality engineering, and to develop critical and complex software systems not only requires a complete, consistent and unambiguous design, and implementation methods, but also a suitable testing environment that meets certain requirements, particularly, to face the complexity issues. Traditional methods, such as analyzing each requirement and developing test cases to verify correct implementation, are not effective in understanding the software's overall complex behavior. In that respect, existing approaches to software testing are viewed as time-consuming and insufficient for the dynamism of the modern business environment. This dynamics requires new tools and techniques, which can be employed in tandem with innovative approaches to using and combining existing software engineering methods. This work advocates the use of a recently proposed software engineering paradigm, which is particularly suited to the construction of complex and distributed software-testing systems, which is known as Agent-Oriented Software Engineering. This methodology is a new one, which gives the basic approach to agent-based frameworks for testing.


*Keywords:* Software testing; Agent; Agent-Oriented Software Engineering; Agent-based software testing; Distributed software testing

## 1. Introduction

Today's innovative software implementation technologies enable the creation of more sophisticated and flexible applications. But they also increase the expertise needed to achieve or extend software quality goals. So, building high quality software for real world applications is difficult. This is true, irrespective of models and techniques are applied for building the software and this is due to the consequence of the essential complexity of software [12]. The expectation is that the next generation of software systems will get even more complex in the aspects of development and testing [10,12].

Testing activities support quality assurance by executing the software being studied to gather information about the nature of the software. The primary advantage of testing is that the software being developed can be executed in its appropriate environment and the results of these executions with the test cases provide confidence that the software will perform as intended and must satisfy the users' requirements [16]. These issues make the software testing a complex process and the primary source of complexity on testing of software systems is the intrinsic complexity of the software being tested.

It is now widely accepted within the software engineering community that the complexity underlying the software engineering processes can be managed through the application of various elementary mechanisms. Booch [1] identifies these mechanisms to be: *abstraction, decomposition, and organization.* The integration of these mechanisms to deliver a coherent approach to engineering software systems can

* Corresponding author.
*E-mail addresses:* pd_chelvoume@yahoo.co.in (P. Dhavachelvan), gvuma@annauniv.edu (G.V. Uma), vskvenkatachalapathy@yahoo.com (V.S.K. Venkatachalapathy).

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2005.12.002

be achieved through the Agent-Oriented Software Engineering (AOSE) paradigm and such integrated frameworks offer much potential to manage the increasing levels of complexity inherent within software engineering processes [11,13].

Agent technologies are interesting to users only if those technologies address the issues of interest to the users. The *Adequacy* and *Establishment* hypotheses [12] support that:

- Agent-oriented decompositions are an effective way of partitioning the problem space of a complex system.
- The key abstractions of the agent-oriented mindset are a natural means of modeling complex systems.
- The agent-oriented philosophy for identifying and managing organizational relationships is appropriate for dealing with the dependencies and interactions that exist in a complex system.

So, from the agents' point of view, the work presented in this thesis is motivated by the hypothesis that agent-based computing offers better approach to manage the complexity that is inherent with the testing of modern software systems.

On the other hand, agents are identified as flexible problem solvers that operate in an environment over which they have only partial control and observability, and whose interactions need to be handled in a similar flexible manner [9,17,8,11]. As agent technology has matured and become more accepted many diverse AOSE approaches and methodologies have been proposed. GAIA (Generic Architecture for Information Availability) [18] and MaSE (Multi-agent Systems Engineering Methodology) [4] are presented as the methodologies for agent-oriented analysis and design. Both the methodologies support the micro-level (agent structure) as well as the macro-level (agent society and organization structure) of agent development. MaSE is similar to GAIA with respect to generality and the application domain supported, but in addition MaSE goes further regarding support for automatic code creation.

SODA (Societies in Open and Distributed Agent spaces), is a methodology proposed for agent-oriented analysis and design of Internet-based applications [15], i.e., this method mainly concentrates on the inter-agent (macro-level) issues. Tropos [2] is proposed to cover early requirements, late requirements, architectural design, detailed design, and implementation. It facilitates the construction of a detailed model of the requirements without design information, which is the significant feature of the Tropos methodology. Prometheus [14] is a methodology aimed at non-experts, which includes three phases: *system specification*, *architectural design and detailed design.* Prometheus functionalities have less implementation bias compared to other methodologies.

Overall, all these methodologies shown that the agent-based computing would seem the most appropriate approach for problem domains, where data, control, expertise or resources are distributed and provide a reasonable support for basic agent-oriented concepts such as autonomy, mental attitudes, pro-activeness, reactiveness, etc. In summary, the reviewed methodologies have shown that there is a concept for analyzing the agent-based systems, but no support for non-agent systems. In addition to that several open issues that include testing, quality assurance, supporting management guidelines, and maintenance are not discussed by any of these methodologies. So from the AOSE perspective, these uncovered issues motivated us to carry out this research on agent-based software testing as an attempt to bridge the gap between software testing and agents.

Based on these two different perspectives, in this paper, we reported on a proposal for agent based approach in software testing. This approach is a first step towards proposing a composite testing environment based on agents, which possibly incorporates several testing techniques and to provide the quantitative justification theory of precisely why agent-based systems are well suited for testing complex systems. The paper is organized as follows: Section 2 defines the Multi-Agent System for software testing as an application domain. Sections 3 and 4 discuss about the implementation issues of different types of agents of the proposed system. Section 5 provides the quantitative statistical analysis over results obtained in the experiments and Section 6 concludes the proposed work with its merits and enhancements.

## 2. Multi-agent system for software testing

**Definition 1.** Let '$S$' be the proposed MAS for providing variety of testing environments (techniques) and it can be defined as

$$S = \left\{ \begin{array}{l} (D_1, D_2, D_3, \ldots, D_z), \\ (a_1, a_2, a_3, a_4, \ldots, a_x), \\ a_1 = (a_1, ac_{11}, ac_{12}, \ldots, ac_{1(K_1-1)}), \\ a_2 = (a_2, ac_{21}, ac_{22}, \ldots, ac_{2(K_2-1)}), \\ \vdots \\ a_x = (a_x, ac_{x1}, ac_{x2}, \ldots, ac_{x(K_x-1)}), \end{array} \right\} \quad (1)$$

where the following hold:

- '$D$' is the *distributor agent* and '$z$' is the number of distributor agents in the MAS.
- '$a_v$' is the *testing agent* and '$x$' is the number of testing agents in the MAS and then $0 < v \leqslant x$. Since this multi-agent framework provides, *scalar type testing environment*, i.e., one agent can provide the testing environment with only one testing technique, '$x$' is the number of testing agents and also the number of testing techniques available in the MAS.
- '$ac$' refers to the *cloning agent*($s$) and '$ac_{vw}$' is the one of the clone of '$a_v$' and then $0 < \mathrm{w} \leqslant k_v - 1$.