Structured Acceptance Test Suite Generation Process for Multi-Agent System

Belkacem Athamena (Corresponding author)
MIS Department, College of Business Administration
Al Ain University of Science and Technology
PO box 64141, Al Ain, Abu Dhabi, UAE
E-mail: athamena@gmail.com

Zina Houhamdi
Software Engineering Department, College of Engineering and IT
Al Ain University of Science and Technology
PO box 64141, Al Ain, Abu Dhabi, UAE
E-mail: z_houhamdi@yahoo.fr

Received: November 13, 2011 Accepted: November 22, 2011 Published: January 1, 2012
doi:10.5539/cis.v5n1p55 URL: http://dx.doi.org/10.5539/cis.v5n1p55

Abstract

In recent years, Agent-Oriented Software Engineering (AOSE) methodologies are proposed to develop complex distributed systems based upon the agent paradigm. The implementation for such systems has usually the form of Multi-Agent Systems (MAS). Testing of MAS is a challenging task because these systems are often programmed to be autonomous and deliberative, and they operate in an open world, which requires context awareness. In this paper, we introduce a novel approach for goal-oriented software acceptance testing. It specifies a testing process that complements the goal oriented methodology *Tropos* and strengthens the mutual relationship between goal analysis and testing. Furthermore, it defines a structured and comprehensive acceptance testing process for engineering software agents by providing a systematic way of deriving test cases from goal analysis.

Keywords: Agent Testing, Goal-Oriented Testing Methodology, Acceptance Testing, Test Case Generation

1. Introduction

MAS are increasingly taking over operations and controls in enterprise management, automated vehicles, and financing systems, assurances that these complex systems operate properly need to be given to their owners and their users (Nguyen *et al*., 2010). This calls for an investigation of suitable software engineering frameworks, including requirements engineering, architecture, and testing techniques, to provide adequate software development processes and supporting tools. In particular, the very specific nature of software agents makes it difficult to apply existing software testing techniques to them. For instance, agents operate asynchronously and in parallel, which challenges testing and debugging. As a result, testing software agents and MAS seeks for new testing techniques dealing with their peculiar nature. The techniques need to be effective and adequate to evaluate agent's autonomous behaviors and build confidence in them.

The strong connection between requirements engineering and testing is widely recognized (Graham, 2002). First, designing test cases early and in parallel with requirements helps discovering problems early thus avoiding implementing erroneous specifications. Secondly, good requirements produce better tests. Moreover, early test specification produces better requirements because it helps to clarify ambiguities in requirements. The link is so important that considerable effort has been devoted to what is called test-driven (or test-first) development. In such approach, tests are produced from requirements before implementing the requirements themselves (Beck, 2002).

Several AOSE methodologies have been proposed (Henderson-Sellers & Giorgini, 2005; Houhamdi, 2011a). In terms of testing and verification, while some consider specification-based formal verification (Dardenne et *al*., 1993; Fuxman *et al*., 2004; Perini *et al*., 2003), other borrow Object-Oriented (OO) testing techniques, taking

advantage of a mapping of agent-oriented abstractions into OO constructs (Cossentino, 2008; Pavon *et al*., 2005). However, a *structured testing process* for AOSE methodologies is still absent.

In this paper, we propose an acceptance testing process that exploits the link between requirements and test cases. We describe the proposed approach with reference to the *Tropos* software development methodology (Mylopoulos & Castro, 2000) and consider MAS as the target implementation technology.

The remainder of the paper is organized as follows. Section 2 recalls basic elements of the *Tropos* methodology and introduces related work. Section 3 discusses the proposed approach, an acceptance testing process and test suite derivation. An example that illustrates how to derive test suites is presented in Section 4. Finally, Section 5 gives conclusion.

2. Background and Related Works

2.1 Tropos

Tropos is an AOSE methodology that covers the whole software development process. Tropos is based on two key ideas. First, the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation. Second, Tropos covers also the very early phases of requirements analysis, thus allowing for a deeper understanding of the environment where the software must operate, and of the kind of interactions that should occur between software and human agents. Tropos methodology spans five phases (Fuxman *et al.*, 2004; Mylopoulos & Castro, 2000):

(1) Early requirements concerned with the problem understanding by studying an organizational setting where the intended system will operate. The output of this phase is an organizational model which includes relevant actors (representing stakeholders) their respective goals (stakeholder's objectives) and their interdependencies.

(2) Late requirements, where the intended system is described within its operational environment, along with relevant functions (hardgoals) and qualities (softgoals). The intended system is introduced as a new actor. It appears with new dependencies with existing actors that indicate the obligations of the system towards its context as well as what the system expects from existing actors in its environment**.**

(3) Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies. More system actors are introduced. They are assigned to subgoals or goals and tasks (those assigned to the system as a whole).

(4) Detailed design, where behavior of each architectural component is defined in more detail including specification of communication and coordination protocols. Agents' goals, beliefs and capabilities are specified in detail using existing modeling languages like UML or AUML, along with the interaction between them should occur between software and human agents.

(5) Implementation. During this phase, the *Tropos* specification, produced during detailed design, is transformed into a skeleton for the implementation. This is done through a projection from the *Tropos* constructs to those of a target agent programming platform, such as JADE (TILAB, 2011). Recent work on mapping *Tropos* goal model to JADEX programming platform is described in (Penserini *et al*., 2006).

2.2 MAS Testing Type

There are four types of testing: *Agent testing*, *Integration testing*, *System testing* and *Acceptance testing.* The objectives and scope of each type is described as follows:

- *Agent testing*. The smallest unit of testing in agent-oriented programming is an agent. Testing a single agent consists of testing its inner functionality and agent's capabilities to fulfill its goals and to sense and effect the environment (Houhamdi, 2011b).
- *Integration testing*. An agent has been unit-tested; we have to test its integration with existing agents. In some circumstances, we have to test also the integration of that agent with the agents that will be developed and integrated subsequently. Integration testing make sure that a group of agents and environmental resources work correctly together which involves checking an agent works properly with the agents that have been integrated before it and with the "future" agents that are in the course of *Agent testing* or that are not ready to be integrated. This often leads to developing mock agents or stubs that simulate the behaviors of the "future" agents (Houhamdi & Athamena, 2011a).
- *System testing*. Agents may operate correctly when they run alone but incorrectly when they are put together. System testing involves making sure all agents in the system work together as intended. Specifically, one

must test the interactions among agents (protocol, incompatible content or convention, etc.) and other concerns like security, deadlock (Houhamdi & Athamena, 2011b).

- *Acceptance testing*. Test the MAS in the customer execution environment and verify that it meets the stakeholder goals, with the participation of stakeholders.

To the best of our knowledge, there is no work dealing explicitly with testing MAS at the acceptance level, currently. In fact, agent, integration, and system test harnesses can be reused in acceptance test, providing execution facilities. However, as testing objectives of acceptance test differ from those of the lower levels, evaluation metrics at this level (such as metrics for openness, fault-tolerance, and adaptivity) demand for further research. In this work, we are interested by acceptance testing and in next section, we present in detail a testing process model and we discuss how to derive systematically test cases from goal models.

3. Test Suite Derivation

The acceptance test is a final test of the system, to be executed after the software passes all of the more extensive unit tests and has been successfully integrated. The acceptance test will consist of a set of selected tests which demonstrate that the system complies with all requirements, but will not be as exhaustive as the unit and integration testing (Trevor, 2009).

Acceptance test suite derivation takes place at the Late Requirements phase, in parallel with the system analysis. At this stage, we have identified: actors, actors' goals, and dependencies between actors. Actors include stakeholders, identified at Early Requirements phase, and system actors. Stakeholder actors present their intentions to the system actors by goal dependencies: they delegate goals to the system actors. *Stakeholder goals* represent stakeholder objectives and requirements towards the intended system. This type of goal is mainly identified at the early requirements phase of *Tropos*. *System goals* represent system-level objectives or qualities that the intended system has to reach or provide. This type of goal is mainly specified at the late requirements phase of *Tropos* (Dastani *et al*., 2006). In general, these goals represent users' objectives and intentions with regard to the intended system, so the fulfillment of these goals is a pivotal benchmark to the system acceptance. Thus, we will use them as foundations for acceptance test suites.

Acceptance test suite derivation consists of the following steps: For each stakeholder actor identified in the early and late requirements phases, a set of goals that the actor delegates or depends on the system is identified. These goals are analyzed by means of decomposition or contribution analysis; and the results are goal decomposition trees inside system actors. Then, for each of these goals, we have to read the corresponding analysis goal tree to identify the leaf goals of the tree, and finally to create a test suite for each leaf goal (see Figure 1).

The derived acceptance test suites can be used for two distinctive objectives: Refining the analysis model and acceptance test. The first objective is realized during acceptance test suite derivation. By using derived suites to review the specification, one could point out problems with the analysis goal model, such as decomposition, unsatisfiability, ambiguities, implicit assumptions, inconsistencies (e.g., a goal cannot be fulfilled or a hardgoal somehow contributes to a softgoal both positively and negatively), and so forth. Problems pointed out at this stage could substantially reduce development effort, since they can be solved before implementation. On the other hand, the second objective requires the system to be built. At this time, derived test suites are used by the customer to evaluate the delivered system to decide eventually whether the system is ready to be deployed or it needs further improvement.

The analysis of each system actor consists of goal decomposition/contribution trees, in that, goals can be decomposed into sub-goals, and sub-goals are means to achieve or to contribute to the goals. According to the introduced steps, we analyze the goal trees and create a test suite for each leaf goal.

Each test suite contains a set of test cases corresponding to the scenarios identified. The operational and usage scenarios and the fulfillment criteria depend on the problem domain, but they often need agreements from both sides: customer and development team. Both, work together to define these scenarios. Finally, the fulfillment of actor's goals can be reasoned on the basis of the fulfillment of the leaf goals and the goal decomposition/contribution trees.

The basic requirement for the system acceptance (i.e. all the derived test suites are passed) entails that all the goals of all the stakeholder actors are achieved or satisfied.

4. Case Study

To illustrate our approach, we introduce a multi-agent system that is composed of several cleaning agents working at a public garden. This software could be deployed on a physical platform composed of a set of moving

robots. Robots are in charge of keeping the garden clean; agents in the system have to collaborate to optimize their work and be nice with visitors.

Following the guidelines of *Tropos*, we do the early requirements analysis and identify stakeholders' goals associated with Robot agent (see Figure 2).

There are two top softgoals that the stakeholder wants to reach: SG1: minimize-cleaning-expense and SG0: improve-service-quality. To reach the latter, two other sub-goals need to be fulfilled: G1: keep-the-garden-clean and SG2: please-visitors. There could be more goals that the stakeholder wants to achieve, but we consider only these goals to keep the example simple and understandable.

Figure 3 shows the late requirements analysis for Robot. The stakeholder delegates three goals SG1, SG2, and G1 to the multi-agent system under construction. At a high-level view, the system adds two hardgoals: G2: team-work and G3: be-polite in order to reach SG1, SG2, as required. Robots must achieve all the three hardgoals. Based on goal models specified in the first two phases Early and Late Requirements, we identify three leaf goals that give rise to three acceptance test suites, following the steps described in system flowchart (Figure 1). Each test suite can have several test cases. The test scenarios presented in Table 1, Table 2 and Table 3 are abstract, and we keep them so to make our example simple.

For instance, for the scenario of the test case TC1.1, we could specify it as follows: The checking area is a rectangle of 15 x 20 meters. At positions (2,4), (4,8), (10,12), (10,17), we put the following garbage: 1 newspaper at (2,2); 2 plastic glasses at (4,5); dust at (15,15); 1 towel and soft drink at (14,12), within a 0.5 meter circle. The robot is put at position (1,1) and is switched on by a stakeholder. It is left alone, cleaning the area for half an hour. Then, it is switched off by a garden staff.

5. Conclusion

This paper introduced a suite test derivation approach for acceptance testing that takes goal-oriented requirements analysis artifact as the core elements for test case derivation. The proposed process has been illustrated with respect to the *Tropos* development process. It provides systematic guidance to generate test suites from modeling artifacts produced along with the development process. We have discuss how to derive test suites for acceptance test from organizational and system goals. These test suites can be used to refine goal analysis and to detect problems early in the development process.

Similar to object-oriented approaches in which test cases are derived from use-case requirements models, we investigate how to derive test cases from goal-oriented *Tropos* requirements models. Specifically, the proposed methodology contributes to the existing AOSE methodologies by providing:

- A testing process model, which complements the development methodology by drawing a connection between goals and test cases, and,
- A systematic way for deriving test cases from goal analysis.

References

Beck, K. (2002). Test driven development: By example. Boston, MA, USA: Addison-Wesley Longman Publishing.

Cossentino, M. (2008). From Requirements to Code with PASSI Methodology. In Sugumaran, V. (Ed.), *Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications*, 491-512.

Dardenne, A., Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2), 3-50. http://dx.doi.org/10.1016/0167-6423(93)90021-G

Dastani, M., Riemsdijk, M., & Meyer, J. (2006). Goal types in agent programming. In Proceeding of the 17th European Conference on Artificial Intelligence (ECAI'06*)*, *Frontiers in Artificial Intelligence and Applications*, 141, 220-224.

Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., & Traverso, P. (2004). Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 2, 132-150. http://dx.doi.org/10.1007/s00766-004-0191-7

Graham, D. (2002). Requirements and testing: Seven missing-link myths. *IEEE Software*, 5, 15-17. http://dx.doi.org/10.1109/MS.2002.1032845

Henderson-Sellers, B., & Giorgini, P. (2005). *Agent-Oriented Methodologies*. USA**:** Idea Group Publishing. http://dx.doi.org/10.4018/978-1-59140-581-8

Houhamdi, Z. (2011a). Multi-agent system testing: A survey. *International Journal of Advanced Computer Science and Applications*, 2(6), 135-141. http://dx.doi.org/10.3844/jcssp.2011.690.697

Houhamdi, Z. (2011b). Test suite generation process for agent testing. *Indian Journal of Computer Science and Engineering*, 2(2), 272-280. http://www.ijcse.com/docs/IJCSE11-02-02-50.pdf

Houhamdi, Z., & Athamena, B. (2011a). Structured integration test suite generation process for multi-agent system. *Journal of Computer Science*, 7(5), 690-697.

Houhamdi, Z., & Athamena, B. (2011b). Structured system test suite generation process for multi-agent system. *International Journal on Computer Science and Engineering*, 3(4), 1681-1688. http://www.enggjournals.com/ijcse/doc/IJCSE11-03-04-036.pdf

Mylopoulos, J., & Castro, J. (2000). Tropos: A framework for requirements-driven software development. *Information Systems Engineering State of the Art and Research Themes SpringerVerlag*, 261-273.

Nguyen, C., Perini A., & Tonella, P. (2010). Goal oriented testing for MASs. Int. J. *Agent-Oriented Software Eng*., 4, 79-109. http://dx.doi.org/10.1504/IJAOSE.2010.029810

Pavon, J., Gomez-Sanz, J. J., & Fuentes, R. (2005). The INGENIAS Methodology and Tools. In Henderson-Sellers, B., & Giorgini, P. (Eds.), *Agent-Oriented Methodologies,* 236-276.

Perini, A., Pistore, M., Roveri, M., & Susi, A. (2003). Agent-oriented modeling by interleaving formal and informal specification. In Proc. AOSE, 36-52. http://dx.doi.org/10.1007/978-3-540-24620-6_3

Penserini, L., Perini, A., Susi, A., & Mylopoulos, J. (2006). From capability specifications to code for multi-agent software. In proceeding of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), 253-256.

TILAB (2011). Java agent development framework. [Online] Available: http://jade.tilab.com/

Trevor, A. (2009). User acceptance testing. *Silverpath technologies Inc*. http://www.silverpath.com/resources/Silverpath-UserAcceptanceTestingWhitepaper-090203.pdf

Table 1. Test suite derived for goal G1 (Keep the Garden Clean)

Test case	Scenario	Criteria
TC1.1	Given an actual area of the garden (A for short), garbage are placed at specified positions (p1; …; pn), the amount of garbage is (a1; a2; …; an), respectively. The robot must clean this area.	The area will be cleaned in less than *t* minutes.
TC1.2	Area A has garbage that is repeatedly thrown into it in a random manner.	The area is periodically cleaned.
TC1.3	Depending on the time at the garden, area A can be more or less dirty: the amount of garbage is a function of time and position.	Robot adapts its cleaning interval and focal positions.

Table 2. Test suite derived for goal G2 (Team Work)

Test case	Scenario	Criteria
TC 2.1	Agents of robots work together in area A.	The agents do not overlap their cleaning areas.
TC2.2	There is two recharging stations (X1;X2) in A.	There is no conflict with regard to the recharging station.

Table 3. Test suite derived for goal G3 (Be polite)

Test case	Scenario	Criteria
TC3.1	While the cleaning agents are moving or cleaning in area A, there are N humans moving in the area along different directions.	The cleaning agents stop moving/working and nod their heads to say hello when they meet a human.

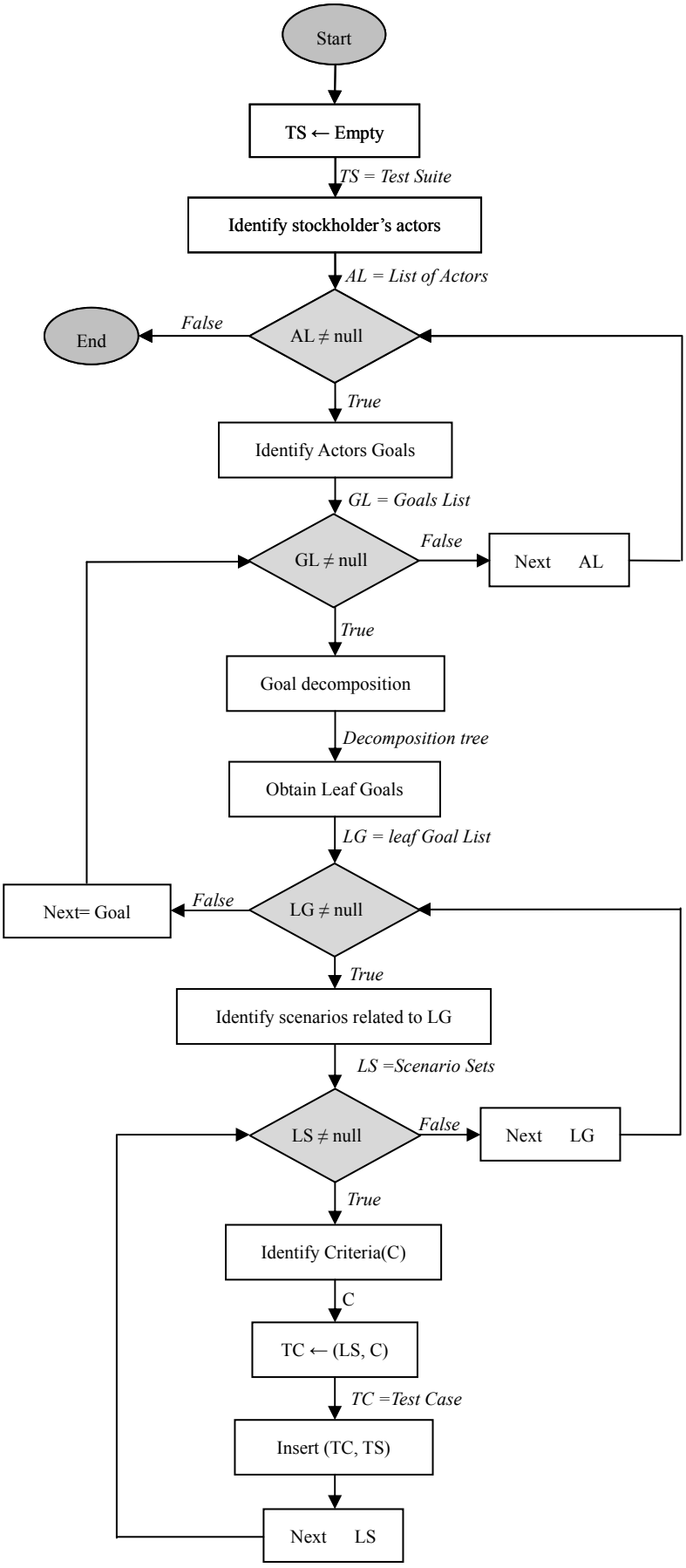

Figure 1. Acceptance test flowchart

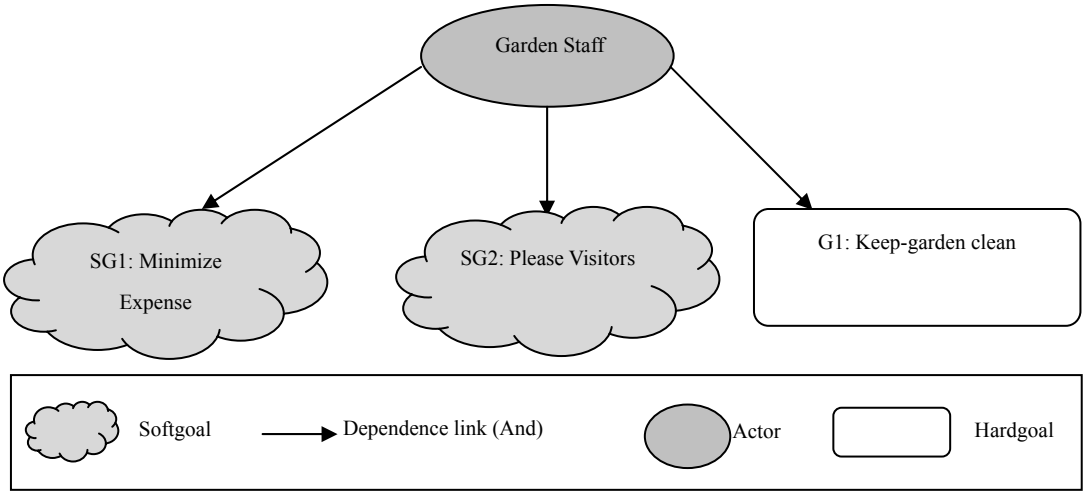

Figure 2. Stakeholder's goals

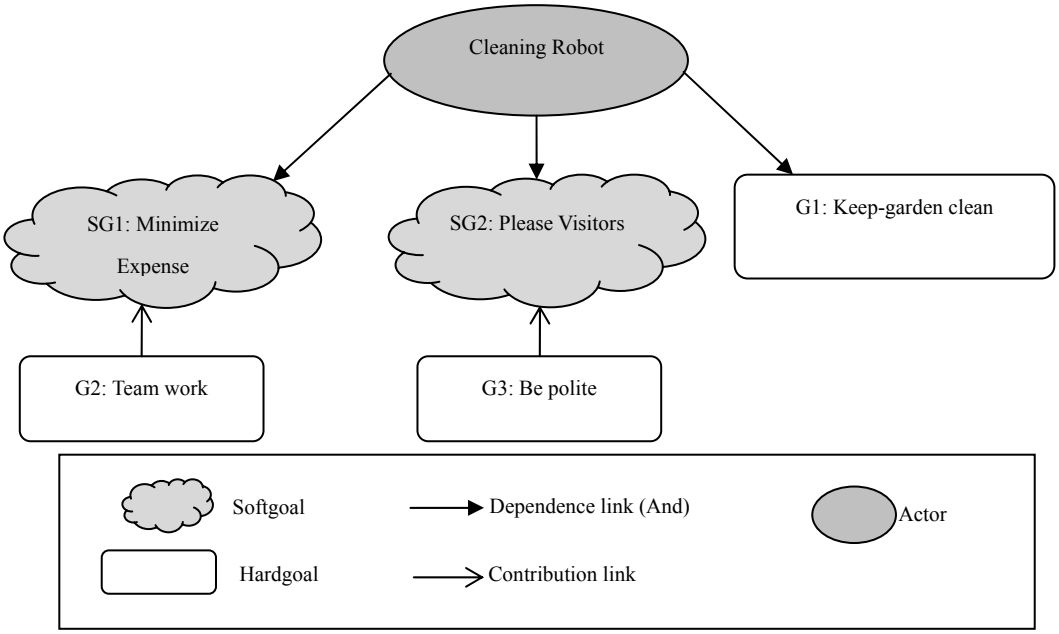

Figure 3. Goal decomposition tree