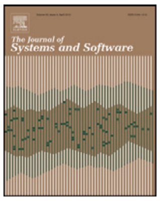
The Journal of Systems and Software 85 (2012) 2770–2784

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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



HPobSAM for modeling and analyzing IT Ecosystems – Through a case study

Narges Khakpour[b,a,e], Saeed Jalili[a,*], Marjan Sirjani[c,d], Ursula Goltz[b], Bahareh Abolhasanzadeh[a]

[a] Tarbiat Modares University, Tehran, Iran
[b] Technical University of Braunschweig, Braunschweig, Germany
[c] Reykjavk University, Reykjavk, Iceland
[d] University of Tehran, Tehran, Iran
[e] Leiden Institute for advanced Computer Science, Leiden University, The Netherland

A R T I C L E I N F O

Article history:
Received 7 May 2011
Received in revised form 7 February 2012
Accepted 7 March 2012
Available online 16 March 2012

Keywords:
Self-adaptive systems
Large-scale software systems
Formal modeling
Verification
Self-organizing systems

A B S T R A C T

The next generation of software systems includes systems composed of a large number of distributed, decentralized, autonomous, interacting, cooperating, organically grown, heterogeneous, and continually evolving subsystems, which we call IT Ecosystems. Clearly, we need novel models and approaches to design and develop such systems which can tackle the long-term evolution and complexity problems. In this paper, our framework to model IT Ecosystems is a combination of centralized control (top-down) and self-organizing (bottom-up) approach. We use a flexible formal model, HPobSAM, that supports both behavioral and structural adaptation/evolution. We use a detailed, close to real-life, case study of a smart airport to show how we can use HPobSAM in modeling, analyzing and developing an IT Ecosystem. We provide an executable formal specification of the model in Maude, and use LTL model checking and bounded state space search provided by Maude to analyze the model. We develop a prototype of our case study designed by HPobSAM using Java and Ponder2. Due to the complexity of the model, we cannot check all properties at design time using Maude. We propose a new approach for run-time verification of our case study, and check different types of properties which we could not verify using model checking. As our model uses dynamic policies to control the behavior of systems which can be modified at runtime, it provides us a suitable capability to react to the property violation by modification of policies.

1. Introduction

The next generation of software systems includes complex systems of systems where the individual systems and components are modeled, built, operated, and controlled by different stakeholders, across organizations. Furthermore, software systems and components are equipped with increasing autonomy, including capabilities for self-configuration and self-organization. We call such systems IT Ecosystems. Such software-intensive IT systems can no longer be designed in a purely centralized fashion. Novel approaches are required to design, develop and analyze these systems.

1.1. Motivation

IT Ecosystems must have the ability to continually evolve and grow even in situations that are unknown during the development time. Due to the fact that it is impossible to fully and properly predict adaptive needs during the design time, adaptive behavior must be built in a way that is *flexible* and modifiable at runtime, because hard-coded mechanisms make tuning and adapting long-run systems complicated.

While each subsystem of a system evolves and changes autonomously to be able to adapt to potentially changing environmental conditions and constraints, they are also cooperating to fulfill a global goal. The centralized control approach to design, in which the behavior of the system is controlled in a top-down way, has attained its limit. In contrast, the decentralized approach relying on a self-organized bottom-up establishment of the desired behavior appears to be infeasible, since we have to make sure that this decentralized approach does not result in unanticipated and undesired behavior. As often, the design of the system has to follow an approach in the middle, somewhere in-between a centralized and a decentralized architecture.

Furthermore, since a complex software system often has a great degree of autonomy, it is more difficult to ensure that it behaves as intended and avoids undesirable behavior. Therefore, to guarantee the functionality of a complex IT Ecosystem, we have to provide mechanisms to ensure that the system is operating correctly, where *model-driven approaches* and *formal methods* can play a key role. Therefore, *we need novel models and approaches to design and develop such systems which can*

* Corresponding author.
E-mail addresses: nkhakpour@modares.ac.ir (N. Khakpour), sjalili@modares.ac.ir (S. Jalili), marjan@ru.is (M. Sirjani), goltz@ips.cs.tu-bs.de (U. Goltz), bahar.abolhasanzadeh@modares.ac.ir (B. Abolhasanzadeh).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2012.03.007

tackle the long-term evolution, flexibility, complexity and assurance problems.

Different frameworks and models have been introduced to design large-scale software systems inspired by natural systems (Villalbaa and Zambonelli, 2011; Beal and Bachrach, 2006; Viroli et al., 2010; Shen et al., 2004). Furthermore, Sloman and Lupu (2010) proposed a flexible policy-based approach for designing ubiquitous systems. Although most of the existing models are able to exhibit properties of self-organization, self-adaptability, and of long-lasting evolvability, they are not provided with a formal foundation. Also, researchers have paid a lot of attention to formal specification and analysis of dynamic adaptation (Bradbury et al., 2004; Taentzer et al., 2000; Cansado et al., 2010; Khakpour et al., 2010a; Zhang and Cheng, 2006). Here, most of the existing approaches deal with either behavioral adaptation or structural adaptation (Becker and Giese, 2008; Bradbury et al., 2004). Adaptation, self-* properties, and autonomous computing are however restricted to responding short-term changes, while systems must be additionally able to evolve and grow to cover the long-term evolution of systems (Deiters et al., 2010).

1.2. Contribution

In this paper we study HPobSAM as a framework for modeling, developing and verifying IT Ecosystem (Khakpour et al., 2010a, in press; Khakpour et al., submitted for publication) illustrated through a transportation service of a smart airport. Our contributions are as follows:

- We illustrate how HPobSAM can be used as a flexible model for designing IT Ecosystems using a transportation service case study.
- We provide an executable formal specification of the model in Maude (Clavel et al., 2003). LTL model checking and bounded state space search are used to analyze the model. We found a cross deadlock and the robot collision in our transportation scenario.
- Due to the complexity of our models, we face state explosion problems when we use model checking to verify some properties. Thus, we employ run-time verification (Lee et al., 1999; Feather et al., 1998) as a complement to model checking in which the executions of the system are monitored and checked against a set of formal specifications. We present a new flexible trace-based approach to verify the system at runtime in which properties to be monitored are specified using an algebra. Then, we transform the algebraic properties into a set of policies which are assigned to an observer. A policy expresses whether an event is expected to occur or not. An observer is modeled as a PobSAM manager that uses the policies to check the conformance of the system behavior to the properties.
- In run-time verification, reaction to the property violations is a main challenge. We address this problem by dynamically defining policies to react to the violations.
- To evaluate the applicability of our approach in practice, we have developed a prototype of our scenario using Java and the Ponder2 tool set (Twidle et al., 2008). We use PonderTalk as the policy language to specify policies.

PobSAM (Policy-based Self-Adaptive Model) (Khakpour et al., 2010a,b) is a flexible formal model for developing and modeling self-adaptive systems which uses policies as the principal paradigm to govern and adapt the system behavior. Policies are known as a powerful mechanism to achieve flexibility in adaptive and autonomous systems which allow us to "*dynamically*" specify the requirements in terms of high level goals. A PobSAM model is a collection of actors, views, and autonomous managers. The autonomous managers govern the behavior of actors by enforcing suitable policies using contextual information provided by views. This model supports behavioral adaptation through modifying the policies used to control the system behavior introduced in Khakpour et al. (2010a). HPobSAM is an extension of this model to support hierarchical modeling and structural adaptation introduced in Khakpour et al. (submitted for publication), in which a manager is aware of its substructure and adapts its substructure to the changing environment according to policies. HPobSAM has a formal foundation that employs an integration of algebraic formalisms and actor-based models. The structural operational semantics of HPobSAM is described using graph transition systems and hierarchical hypergraph transformation. In this paper, we (i) study the applicability of HPobSAM in designing IT Ecosystems, (ii) provide the mapping of HPobSAM to Maude, (iii) propose a new runtime verification approach for HPobSAM, and (iv) introduce the detailed case study of the smart airport and its modeling, verification and implementation using HPobSAM.

In this paper we explain HPobSAM and our analysis techniques through a case study, smart airport. The smart airport case study is introduced in Section 2. After giving an overview of HPobSAM in Section 3, in Section 4 we explain the modeling framework and discuss why HPobSAM is suitable for modeling IT Ecosystems in general. In Section 5 we show the HPobSAM model for the smart airport. In Section 6, we present the Maude specification of our model and analyze the model formally. An approach is proposed to verify our case study at run time in Section 7. We compare our approach with related work in Section 8, and Section 9 concludes the paper.

2. Case study overview

The airport departure scenario is an example of a software-intensive system of systems (Deiters et al., 2010). We use a transportation service at the departure area of an airport as our case study. This transportation service is supposed to be realized by a number of Autonomous Transport Vehicles (ATVs) which are responsible to transport passengers between stopovers including passenger entrances, check-in desks, departure gates, and plane parking positions. There are several two-lane roads which connect the aforementioned stopovers. To avoid congestion and blockages, there are some side roads which can be used instead of the main roads (implying a reduced vehicle speed).

There are a variety of ATVs of different sizes to perform transportation in a self-organizing manner. All ATVs know the airport map and stopovers. The transportation service of the transport vehicles contains transporting passenger (i) from an airport entrance to one of the five check-in desks, (ii) from a check-in desk to one of the departure gates, and (iii) from a departure gate to the correct parking position of the respective plane. ATVs consume energy while driving on roads, and they have to recharge their batteries at a charging station.

The observation systems (e.g. smart cameras, sensors, RFID readers) placed around the area gather and provide information (e.g. the current traffic information). This information is used by ATVs in order to achieve a good performance of transportation. Furthermore, passengers use a mobile device, called SmartFolk, to interact with the IT systems at the Smart Airport. A SmartFolk can be seen as a device like a PDA. Within the IT Ecosystem they represent their owners and act as interfaces to the airport IT Ecosystem.

ATVs are signed in a service named transport scheduler that collects passenger orders and offers tickets (pickup/drop positions, times) to the ATVs. Hence ATVs have to collaborate and negotiate in competition on tickets, roads and charging stations. To prevent the occurrence of unsafe situations caused by a selfish acting ATV, we need to implement a mechanism to balance agent autonomy and system controllability.