Science of Computer Programming 97 (2015) 41–46

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Developing correct, distributed, adaptive software

Mila Dalla Preda, Maurizio Gabbrielli *, Saverio Giallorenzo, Ivan Lanese, Jacopo Mauro

Department of Computer Science, University of Bologna/Lab. Focus INRIA, Italy

HIGHLIGHTS

- We develop a methodology for verifying distributed, adaptive software systems.
- The cornerstone of our framework is the use of choreography languages.
- We integrate also different techniques based on abstract interpretation and on dynamic verification.

ARTICLE INFO

Article history:
Received 8 October 2013
Accepted 5 November 2013
Available online 13 November 2013

Keywords:
Choreography languages
Adaptation
Abstract interpretation

ABSTRACT

We illustrate our approach to develop and verify distributed, adaptive software systems. The cornerstone of our framework is the use of choreography languages, which allow us to obtain correctness by construction. Workflow Patterns are also used as abstract tools to design real systems, while techniques based on abstract interpretation and on dynamic verification are integrated in our framework to reduce the complexity of verification.

It has been a pleasure to write this paper for the Paul's festschrift for three reasons. The first one is that I met the first time Paul many years ago, when I was a post-doc at CWI. Even though our interests were different, I had the occasion to meet him occasionally and to appreciate his work. The second reason is that I had the opportunity to appreciate the professional skills and the human qualities of Paul in the context of the board of EAPLS, where his work has been essential for the life of the organization. The third reason is that writing this paper I discovered some old work of Paul which is still relevant today, also for our work. Happy Birthday Paul, also on behalf of my co-authors!
Maurizio

1. Introduction

Adaptive, distributed software has applications in many domains and systems, exhibiting deeply different characteristics. Long running (often distributed) systems live for long periods of time and therefore should adapt to varying contextual conditions, user requirements and execution environments. More importantly, the details of the adaptation needs, and the solutions to be used to answer them, may not be known when the system is designed, deployed or even started. Such systems thus need to dynamically adapt their behavior (this can be autonomic, or may require an external intervention). Particularly important in this context are mission critical Adaptive Control Systems used in Cyber-Physical systems to respond to changes in the physical environment.

Development and verification of distributed, adaptive systems pose several formidable challenges. First, the current development technology is not well suited to develop and verify large-scale adaptive distributed systems due to the lack of

* Corresponding author.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.11.019

high-level structuring abstractions for complex communication behavior or for context-aware adaptation. In recent years, session types [10–12,23], choreography languages [1,29], behavioral contracts [6], and ad-hoc scripting languages [2] have been advocated as possible abstractions for providing high-level specifications describing the expected behavior of a distributed system. These concepts usually consider static techniques, which alone are not sufficient to model dynamically adaptable software. Indeed, the assumption on either the ability to type-check the component source code, or the availability of its complete behavioral interface, may not be realistic in presence of adaptation. Particularly relevant in this context are *Interaction-Oriented Choreographies* (IOCs) [29] which allow to abstractly describe the participants to a distributed protocol, the interactions among them, and their order. From an IOC one can automatically generate a detailed description of the behavior of each participant, expressed in terms of a *Process-Oriented Choreography* (POC), which, in many cases, provides an executable code. A main result in this setting is that the POC automatically derived from a given IOC correctly implements the behavior specified by the given IOC, inheriting relevant correctness properties such as deadlock freedom [29]. IOCs and their projection onto POCs have been studied in different contexts [10,12,23,29], and integrated into different kinds of languages [11,33]. A relevant limitation affecting all the IOC-based approaches is that they can be applied only to systems whose structure is static and fully known since the very beginning.

Concerning more specifically adaptive systems, several middleware and architectures enabling run-time adaptation have been proposed in the literature, such as [7,13,18,28,36] (an interesting survey can be found in [30]). Other approaches instead rely on traits [22], role-based modeling [21], and aspect-oriented programming [35]. While these approaches provide tools for programming adaptive systems, the challenge of ensuring that those systems behave as expected after the execution of some adaptation steps is still open. One cannot know a priori the structure of the adapted system, and this seems to make the use of static analysis techniques impossible. For this reason, most of the approaches in the literature offer no guarantee on the behavior of the adaptive system after adaptation [7,13,28], or they assume to know all the possible adaptations in advance [36].

Verification of adaptive systems is very difficult also because of the combinatorial effect due to the composition of different variants of the individual modules and to the run-time nature of system configuration. Many aspects of such composition and configuration can only be partially foreseen at design and compile time, hence performing static checks would require to consider a very large number of possibilities, including many which will never happen at run-time. A possible solution here could be to move a part of the static analysis to execution time, along the lines of what happens, for example, with the analysis of bytecode that the JVM performs at class loading time.

1.1. Our approach

In order to attack the challenges mentioned above we intend to develop effective methodologies and tools for proving correctness of adaptive software systems. Our approach is based on the integration of different techniques, including abstract interpretation, choreography languages, and design patterns. More precisely, we aim to reach the following objectives:

1. To define a framework which allows us to statically prove correctness properties of adaptive distributed systems, such as deadlock freedom and termination, by using choreography languages.
2. To devise suitable design patterns which, exploiting the correctness results of our framework, allow us to construct correct, distributed, adaptive software.
3. To include in such a framework suitable abstract interpretation techniques to reduce the complexity of verification of real systems.
4. To develop a formal theory based on choreography languages and abstract interpretation for statically proving security properties of adaptive systems (e.g., non-interference).
5. To integrate the static techniques with run-time monitoring and, more generally, dynamic verification techniques.

The rest of this paper is devoted to illustrate in more detail these objectives. For some of them, notably the first and the second, we have already several results [15,17], while for others our research is at an early stage and we expect to have results in the medium-long term.

2. Correctness by construction: choreography languages

Correctness properties of adaptive systems can be imposed *by design*, by using choreography languages. Correctness by design is obtained by automatically deriving executable code for adaptive distributed systems from high-level, IOC-like specifications. Our technique [15] is based on a careful split of the system specification into a description of the initial system, to be checked before deployment, and a description of the adaptation steps, each of which can be defined and checked in isolation, even while the system is running.

We use a *rule-based approach* to adaptation which relies on the following architectural model: the adaptive system is composed by interacting participants deployed on different locations, each executing its own code and accessing its own local state. Adaptation is performed by an *adaptation middleware*. The middleware includes one or more, possibly distributed, *adaptation servers*, which are repositories of *adaptation rules*. Adaptation rules can be added or removed at any moment,