Science of Computer Programming 94 (2014) 333–361

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Dynamic adaptation with distributed control in Paradigm

S. Andova [a], L.P.J. Groenewegen [b], E.P. de Vink [c,d,*]

[a] ICT–Technology, Fontys University of Applied Sciences, Eindhoven, The Netherlands
[b] Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
[c] Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands
[d] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 31 October 2011
Received in revised form 14 November 2013
Accepted 19 November 2013
Available online 26 November 2013

Keywords:
Component-based systems
Dynamic system adaptation
Distributed control
Formal verification

ABSTRACT

Adaptation of a component-based system can be achieved in the coordination modeling language Paradigm through the special component McPal. McPal regulates the propagation of new behavior and guides the changes in the components and in their coordination. Here we show how McPal may delegate part of its control to local adaptation managers, created on-the-fly, allowing for distribution of the adaptation indeed. We illustrate the approach for the well-known example of the dining philosophers problem, by modeling migration from a deadlock-prone solution to a deadlock-free and starvation-free solution without any system quiescence. The system migration goes through various stages, exhibiting a shift of control among McPal and its helpers, and changing degrees of orchestrated and choreographic collaboration. The distributed system adaptation is formally verified using the mCRL2 model checker.

1. Introduction

Many systems today are affected by changes in their operational environment when running, while they cannot be shutdown to be updated and restarted again. Instead, *dynamic adaptive systems* must be able to change their behavior on-the-fly and to self-manage adaptation steps accommodating a new policy.

Dynamic adaptive systems consist of interacting components, usually distributed, and possibly hierarchically organized. In such a system, components may start adaptation in response to various triggers, such as changes in the underlying execution environment (e.g. failures or network congestion) or changes of requirements (e.g. imposed by the user). Adaptation of one component in the system may inadvertently influence the behavior of the components it is interacting with, possibly bringing about a cascade of dynamic changes in other parts of the system. Therefore, the adaptation of a distributed system is a combination of local changes per component and global adaptation across components and hosts in the system. As such, adaptation has to be performed in a consistent and coordinated manner so that the functionality of each separate component and of the system as a whole are preserved while the adaptation is in progress. Due to the complexity of the distributed dynamics of a system adapting on-the-fly, it may be rather difficult to understand whether a realization of a change plan indeed allows the system to perform as it is supposed to, and does not violate any of its requirements, during and after system adaptation.

One way to circumvent this is to formally model and analyze the system behavior and the adaptation changes to be followed. In [28,5,8] we advocated how orchestrated adaptation can conveniently be captured in the coordination modeling language Paradigm (backed up by a translation into the specification languages of the model checkers mCRL2 and Prism).

* Corresponding author at: Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands.
E-mail address: e.p.d.vink@tue.nl (E.P. de Vink).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.11.034

In Paradigm, a system architecture is organized along specific collaboration dimensions, called partitions. A partition is a well-chosen set of sub-behaviors of the local behavior of a component, specifying the phases the component goes through when taking part in the collaboration. At a higher layer in the architecture, the component participates via its role, an abstract representation of the phases. A protocol determines the phase transfers for the components involved. In fact, in Paradigm, dynamic adaptation is modeled as just another collaboration, coordinated by a special component *McPal* [28,5,8]. As progress within a phase is completely local to the component, the use of phase transfer instead of state transfer, is the key concept of Paradigm. This makes it possible to model, at the same time and separated from one another, both behavioral local changes per component, and global changes across architectural layers.

Within Paradigm, the possibility of splitting up coordination and cooperation between components within a system architecture into collaborations, each following its own protocol, has useful consequences for Paradigm's applicability and scalability. In a context without adaptation, Paradigm models have been developed and successfully applied for quite large coordination situations [43,39,45,46]. In [43] the latest UNIX version publicly available at that time, is remodeled and studied for a multi-processor platform. In [39] the CRIS-case from Information Systems is modeled and studied as well as some elevator systems. In [45,46] different architectures of hospital information systems are modeled and investigated. The modeling invariably is in Paradigm, the investigations are through arguing and simulation, as at that time a connection with formal verification was not established yet. These and other examples underpin both applicability and scalability of Paradigm modeling, based on clearly separated collaborations. All this is without formal verification as well as without adaptation. Main lessons learned are, Paradigm models for large real world systems can be developed; such large systems may range from purely technical ICT (software) to purely human (business).

From 2006 on, we have published about *McPal* [28,5,8]: how a Paradigm model, through a special component called *McPal*, can adapt itself into a new Paradigm model, on-the-fly of the model's ongoing dynamicity. In [28] a rather rigid *McPal* is used for two consecutive adaptations: from a not so efficient non-deterministic solution for three components competing for a critical section, via a more efficient non-deterministic solution (with a "smaller" critical section), finally to a deterministic round robin solution (for the smaller critical section). In [5] the *McPal* component is rather more flexible as it can adapt itself too. Moreover, the migration trajectories are formally verified. The migration example is similar to the [28] example, but the migration is more direct: from the not so efficient non-deterministic solution in one migration to the more efficient deterministic round robin solution. Model checking is done with `mCRL2`. Some variants of *McPal* are discussed too. In [8] the same *McPal* is used to migrate from a deterministic round robin solution for a 4-party critical section problem to a probabilistic solution. Here *McPal* is achieving this via gradually adding more probabilistic control into the solution at the cost of the original round robin character. Model checking is done via Prism instead of via `mCRL2`. From 2008 on, papers [7,9,5,8,10] explain how to translate a Paradigm model into the `mCRL2` language. Main lessons learned are, for Paradigm models self-adaptation is a lazy or just-in-time form of coordination in Paradigm style. Moreover, as Paradigm models can be translated to process algebra, model checking of Paradigm models can be carried out. In particular, migration trajectories arising in self-adapting Paradigm models can be formally analyzed through model checking.

Both with respect to migration and with respect to model checking, applicability and scalability are substantially lagging behind the applicability and scalability of modeling in Paradigm without adaptation and without additional model checking. For Paradigm modeling with model checking we have made some progress in scalability in [10], as roles and the clear separation of collaborations at a higher level do facilitate reduction in the state space to be model checked. For the adaptation we have only a first beginning of understanding the model suites (sequences of consecutive Paradigm models appearing and disappearing one after another) during the migration trajectories that can be realized by the adaptation.

The split-up in collaborations adds to the scalability of the approach: protocols, capturing component interaction, that are orthogonal to each other can be addressed more focusedly, namely with respect to the specific roles of the components involved. By focusing on one specific collaboration at a time only part of the detailed behavior of the component is needed explicitly, which should lead to smaller state spaces to be built. Further work is needed to assess the potential benefits, in particular when components are intertwined in many different ways. As in the remaining sections of this paper delegation of migration coordination is structured according to clearly separated collaborations, we have the impression this is useful for scalability and so for applicability of the *McPal* approach, as it is in line with the reasons underpinning Paradigm's applicability and scalability for normal coordination, see [43,39,45,46] and the above remarks concerning these papers.

The suitability of Paradigm to model distributed adaptation strategies, extending our earlier centralized adaptation studies, is discussed in this paper on the dining philosophers example (see also [4]). A deadlock-prone solution of the dining philosophers problem is taken as a source system, to be migrated to a target solution, both deadlock-free and starvation-free. Both systems are modeled in Paradigm, as is the migrating from source to target. As typical for dynamic adaptation in Paradigm, *McPal* regulates the propagation of new behavior and guides the structural changes in the components and in their coordination. Seamlessly, parallel to ongoing execution, new behavior is woven into the ongoing behavior, such that smooth migration is established from *as-is* behavior (of the source system) via intermediate behavior to *to-be* behavior (of the target system). Note, both intermediate and *to-be* behaviors were originally unknown. All this is characteristic for *McPal*. But here, although adding to the complexity of the solution, *McPal* delegates part of its control to local adaptation managers $McPhil_i$, $i = 1, \ldots, 5$, one for each philosopher, while *McPal* keeps controlling them globally. Thus, we argue, the component-based character of the Paradigm language allows for modeling distributed adaptation: separate modeling of strategies for changes of local behavior, coordinated by *McPal* as system adaptation manager guiding the architectural changes. This re-