The Journal of Systems and Software 91 (2014) 24–47

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Dynamic adaptation of service compositions with variability models

G.H. Alférez[a,*], V. Pelechano[b], R. Mazo[c], C. Salinesi[c], D. Diaz[c]

[a] Facultad de Ingeniería y Tecnología, Universidad de Montemorelos, Apartado 16-5, 67500 Montemorelos, Mexico
[b] Centro de Investigación en Métodos de Producción de Software (ProS), Universitat Politècnica de València, Camí de Vera s/n, E-46022 Valencia, Spain
[c] CRI, Panthéon Sorbonne University, 90 rue de Tolbiac, 75013 Paris, France

ARTICLE INFO

Article history:
Received 13 November 2012
Received in revised form 6 June 2013
Accepted 17 June 2013
Available online 25 June 2013

Keywords:
Variability
Models at runtime
Autonomic computing
Dynamic adaptation
Dynamic software product line
Web service composition
Constraint programming
Verification

ABSTRACT

Web services run in complex contexts where arising events may compromise the quality of the whole system. Thus, it is desirable to count on autonomic mechanisms to guide the self-adaptation of service compositions according to changes in the computing infrastructure. One way to achieve this goal is by implementing variability constructs at the language level. However, this approach may become tedious, difficult to manage, and error-prone. In this paper, we propose a solution based on a semantically rich variability model to support the dynamic adaptation of service compositions. When a problematic event arises in the context, this model is leveraged for decision-making. The activation and deactivation of features in the variability model result in changes in a composition model that abstracts the underlying service composition. These changes are reflected into the service composition by adding or removing fragments of Business Process Execution Language (WS-BPEL) code, which can be deployed at runtime. In order to reach optimum adaptations, the variability model and its possible configurations are verified at design time using Constraint Programming. An evaluation demonstrates several benefits of our approach, both at design time and at runtime.

1. Introduction

Software is executed in complex, heterogeneous, and highly intertwined computing infrastructures in which a diversity of events may arise. For example, security threats, network problems, performance reduction in one of the servers, etc. In these situations, it is desirable to adapt the software to continue offering the required functionality. *Software adaptation* can be seen as the ability for humans to reconfigure the software and then restart it, or the ability of the software to reconfigure itself during execution (Akkawi et al., 2007). The first case can be referred to as *static adaptation* and the second one as *dynamic adaptation*. It is possible to carry out static adaptations in cases where the system can be shut down in order to make the required manual adaptations. However, there are critical systems that cannot be stopped to implement the adaptations, e.g. software that run power grids and software for global banking. In such cases, software needs to dynamically adapt its behavior at runtime in response to changing conditions in its supporting computing infrastructure (McKinley et al., 2004; Cetina et al., 2009; Alférez and Pelechano, 2011a). *Dynamic adaptation* of software behavior refers to the act of changing the behavior of some part of a software system as it executes, without stopping or restarting it (Keeney, 2004).

In order to carry out dynamic adaptations, we argue that software needs to take the following key issues into account:

- ***Context Awareness:*** For the purpose of supporting dynamic adaptations, software should be aware of changes in its context. The *context* is any information that can be used to characterize the situation of an entity (Dey, 2001). Context-aware systems are concerned with the acquisition of context, the abstraction and understanding of context, and application behavior based on the recognized context (Schmidt, 2002).
- ***Adaptation Policies:*** *Adaptation policies* change the behavior of the system during execution (Morin et al., 2008). They state in a declarative manner the actions required to adapt the running system to a configuration that better fits its current context.
- ***A Supporting Infrastructure:*** It is unthinkable to depend on manual adaptations because of the inherent intricacy of today's systems and the desired prompt responses. Furthermore, critical systems cannot be stopped in order to carry out the necessary adaptations. Thus, a computing infrastructure should provide support for dynamic adaptations to face context events (Alférez and Pelechano, 2011a; Cetina et al., 2009).
- ***Verification:*** Adapting the system according to changes in the context is not enough. It is necessary to ensure that new system configurations are not invalid in a given situation.

* Corresponding author. Tel.: +34 635215647.
E-mail addresses: harveyalferez@um.edu.mx, harveyalferez@gmail.com (G.H. Alférez), pele@dsic.upv.es (V. Pelechano), raulmazo@gmail.com (R. Mazo), camille.salinesi@univ-paris1.fr (C. Salinesi), daniel.diaz@univ-paris1.fr (D. Diaz).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.06.034

1.1. The need for dynamic adaptation of service compositions

A good example of systems that require dynamic adaptations are the ones based on Web service compositions (or simply called service compositions). Web services have evolved as a standardized and technology-agnostic interoperable way of integrating processes and applications (Little, 2003). Basically, a *Web service* is a special software component that is searched, bound, and executed at runtime and allows systems to interact through standard Internet protocols (Koning et al., 2009). In order to reach the full potential of Web services, they can be combined to achieve specific functionalities. If the implementation of a Web service business logic involves the invocation of other Web services, it is called a *composite service*. The process of assembling a composite service is called *service composition*.

Web services run in a context (e.g. their operating computing infrastructure). In an ideal scenario, Web service operations would do their job smoothly. However, several exceptional situations may arise in the complex, heterogeneous, and changing contexts where they run. For instance, a Web service operation may have greatly increased its response time or may have become unavailable. Cases like these make evident the need for dynamic adaptations in critical systems that are based on service compositions. These adaptations may be triggered in order to do the following at runtime: keep certain contracts known as service-level agreements (SLAs), offer extra functionality depending on the context, protect the system, or make the system more usable.

Related work about dynamic adaptation of service compositions can be classified into three groups. The first group supports dynamic adaptation at the language level (Colombo et al., 2006; Koning et al., 2009; Baresi and Guinea, 2011; Narendra et al., 2007; Sonntag and Karastoyanova, 2011; Moser et al., 2008). This approach can hinder reasoning about adaptations with complex and error-prone scripts (Fleurey and Solberg, 2009). The second group focuses on low-level implementation mechanisms for self-adaptation (Erradi and Maheshwari, 2005; Cardellini et al., 2010; Mosincat and Binder, 2008). This approach lacks support for analyzing the inherent variability of dynamic adaptation at design time. The third group deals with modeling variability in service compositions that support business processes (BPs) (Nguyen et al., 2011; Sun et al., 2010; Hadaytullah et al., 2009; Razavian and Khosravi, 2008; Rosemann and Van der Aalst, 2007; Gottschalk et al., 2008; Puhlmann et al., 2005). Research works in this group propose the creation of variability models that are only used at design time. We argue that the knowledge in variability models should be leveraged at runtime to guide adaptations and hide the complexity of the adaptation space. Moreover, the approaches in the aforementioned groups lack verification of possible service composition configurations caused by dynamic adaptations.

1.2. Contribution

In this paper, we propose a framework that uses variability models at runtime to support the dynamic adaptation of service compositions. This framework spans over design time and runtime, and states the models, tools, and artifacts that can be used to support dynamic adaptations.

At design time, the framework provides tool-supported steps for creating the models that guide autonomic changes. In general terms, a service composition can be viewed as the assembly of pieces to deliver functionality; those pieces can be Web services offered by different providers or composite services themselves. We argue that in the advent of problematic events, functional pieces can be added, removed, replaced, split or merged from a service composition at runtime, hence delivering a new service composition configuration. To this end, we propose that service compositions be abstracted as a set of features in a variability model. A *feature* can be defined as "a logical unit of behavior specified by a set of functional and non-functional requirements" (Bosch, 2000). Thus, adaptation policies describe the dynamic adaptation of a service composition in terms of the activation or deactivation of features in the causally connected variability model (i.e., changes in this model cause the service composition to adapt and vice versa). The variability model and its possible configurations are verified by means of Constraint Programming (CP) prior execution to ensure safe recompositions.

At runtime, a computing infrastructure detects problematic events that arise in the context and carries out the necessary adjustments on the service composition. The activation and deactivation of features in the variability model result in changes in a composition model, which is an abstract representation of the underlying service composition. These changes are reflected into the service composition by adding or removing fragments of Business Process Execution Language (WS-BPEL) code, which are deployed at runtime. WS-BPEL is a standard language for specifying BP behavior based on Web Services (OASIS, 2007). Flexible service composition updates are possible through Dynamic Software Product Line (DSPL) engineering (Hallsteinsen et al., 2008).

This paper offers several novel contributions beyond those of our previously published works (Alférez and Pelechano, 2011a,b, 2012b; Cetina et al., 2009): (1) in our previous work, the generation of variability model configurations was carried out manually. This was an error-prone and time-consuming task. As a result, in this paper we propose a tool that automatizes the creation of variability model configurations at design time; (2) our previous work did not support the verification of the variability model and its possible configurations. As a result, some undesirable dynamic adaptations could emerge at runtime. Therefore, in this paper we incorporate a constraint logic programming solver to automatize the verification of the variability model and its configurations. Although this solver has been used in our previous work (Mazo et al., 2012), it is the first time that it is used to verify the variability model configurations that are used at runtime; (3) context reasoning has been further exploded by means of inference rules and more expressive adaptation policies; (4) we propose an strategy to avoid the saturation of the system when several context events arise in tight time frames; (5) in our previous work, the translation of changes in the composition model into WS-BPEL code was slow (Torres et al., 2012). In this paper, we propose a faster solution based on fragments of WS-BPEL code; and (6) in this paper, our approach supports dynamic adaptations on an enterprise orchestration engine. Instead of extending the functionality of the orchestration engine, we offer a transparent solution in which the engine does not have to be modified.

The remainder of this paper is structured as follows: Section 2 describes a running example that illustrates the need for dynamic adaptation of service compositions. Section 3 gives an overview of our framework for dynamic adaptation of service compositions. Section 4 describes the models that are created at design time to support dynamic adaptations. Section 5 describes the computing infrastructure to deal with dynamic adaptations. Section 6 introduces a demonstration of our framework. Section 7 presents the evaluation of our framework. Section 8 presents related work, and Section 9 presents conclusions and future work.

2. Running example

To illustrate the need for self-adaptive service compositions, we introduce a composite service that supports online book shopping at Orange Country Bookstore. The example is specified with the Business Process Model and Notation (BPMN) in Fig. 1. BPMN tasks express Web service operations (e.g. UPS Shipping service);