Science of Computer Programming 78 (2013) 1035–1054

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

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

"ADORE", a logical meta-model supporting business process evolution

Sébastien Mosser [a,b,*], Mireille Blay-Fornarino [c]

[a] Inria Lille–Nord Europe, LIFL CNRS UMR 8022, University of Lille 1, France
[b] SINTEF IKT, Oslo, Norway
[c] I3S CNRS UMR 7271, University of Nice, France

ARTICLE INFO

Article history:
Received 11 April 2011
Received in revised form 3 January 2012
Accepted 19 June 2012
Available online 20 July 2012

Keywords:
SOA
Business processes
Sep. of concerns
Logical composition

ABSTRACT

The Service Oriented Architecture (SOA) paradigm supports the assembly of atomic services to create applications that implement complex business processes. Since "real-life" processes can be very complex, composition mechanisms inspired by the *Separation of Concerns* paradigm (*e.g. features, aspects*) are good candidates to support the definition and the upcoming evolutions of large systems. We propose ADORE, *"an Activity meta-moDel supOrting oRchestration Evolution"* to address this issue. The ADORE meta-model allows process designers to express in the same formalism business processes and *fragments* of processes. Such *fragments* define additional activities that aim to be integrated into other processes and adequately support their evolution. The underlying logical foundations of ADORE allow the definition of interference detection rules as logical predicate, as well as the definition of consistency properties on ADORE models. Consequently, the ADORE framework supports process designers while they design and then apply evolutions on large processes, managing the detection of interferences among fragments and ensuring that the composed processes are consistent and do not depend on the order of the composition.

1. Introduction

In the Service Oriented Architecture (SOA) paradigm an application is defined as an assembly of services that typically implements mission-critical business processes [1]. Services are loosely coupled by definition, and complex services are built upon basic ones using composition mechanisms. These compositions describe how services will be orchestrated when implemented, and are created by business process specialists. In this context, a large part of the intrinsic complexity of an application is shifted into the definition of business processes. To perform such a task, process designers use "*composition of services*" mechanisms (*e.g.*, orchestrations of services) and realize their use cases as message exchanges between services. A factor that contributes to the difficulty of developing complex SOA applications is the constant evolution of the underlying business. As a consequence, a business expert initially designs its own process, which is then enriched by evolution added by other experts, *e.g.*, the *security* expert, the *persistence* expert, and the *sales* expert.

This situation emphasizes the need for *separation of concerns* (SoC) mechanisms as a support to the design and the evolution of complex business processes. In the SoC paradigm, concerns are defined separately, and finally assembled into a final system using composition techniques. In the context of SOA, it leads us to compose orchestrations according to the upcoming evolutions, that is, to compose "*composition of services*". The complexity of building the final behavior that integrates all the concerns together is delegated to an algorithm.

* Corresponding author at: SINTEF IKT, Oslo, Norway.
E-mail addresses: sebastien.mosser@sintef.no, sebastien.mosser@gmail.com (S. Mosser), blay@polytech.unice.fr (M. Blay-Fornarino).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.06.009

The contribution of this paper is to support the evolution of business processes through a compositional approach, *i.e.*, to automatically support at the model level the evolution of complex business processes through the composition of smaller artifacts. We propose Adore to fulfill this goal. Adore means *"an Activity meta-moDel supOrting oRchestration Evolution"* and supports a compositional approach for process evolution. Evolutions are reified as models describing smaller orchestrations of services to be composed to produce the evolved system.

2. Running example: exploring pictures folksonomies

To illustrate this work, we use the PicWeb system. This example was initially designed for a M.Sc. lecture focused on *"Workflow & Business Processes"*, given in two universities (*i.e.*, the Polytech'Nice School of Engineering and the University of Lille 1). As a consequence, it follows Soa methodological guidelines, positioning it as a typical usage of a Soa for experimental purposes [2]. PicWeb is a subpart of a larger legacy system called jSeduite,[1] which delivers information to several devices in academic institutions. It supports information broadcasting from external partners (*e.g.*, transport network, school restaurant) to several devices (*e.g.*, smart-phone, PDA, desktop, public screen). jSeduite is currently deployed and used daily in three institutions (*i.e.*, Polytech'Nice, Ies Clément Ader and Irsam), where large plasma screens are used to broadcast public information. PicWeb is a subpart of jSeduite, retrieving a set of pictures annotated with a given *tag* from existing partner services. It implements a business process called `getPictures`. Based on a received *tag*, the process retrieves all the available pictures associated with *tag* from the usual repositories (*e.g.*, flickr, provided by Yahoo!). In the remaining paragraphs, we assimilate PicWeb into this particular business process. We represent in Fig. 1(a) the first implementation of PicWeb, as initially defined in jSeduite. This process (i) receives a `tag` (a_0), (ii) asks a registry for the Api key associated to Flickr® (a_1), (iii) retrieves the relevant pictures from the repository (a_2) and finally (iv) replies this set to the user (a_3).

Need for evolutions in PicWeb. Soa systems are highly versatile, as they need to be continuously adapted to their underlying business. We illustrate this need in PicWeb through the change history of the system. These evolutions were driven by changes in the business needs, impacting the business process. For example, flickr conditions changed to restrict the number of times its service can be called per day. The business process had to evolve (*e.g.*, introducing a cache to reduce call frequency) to tackle this unforeseen change. Each evolution implies to change an activity graph increasingly complex, enforcing the need for SoC mechanisms. The final process represented in Fig. 1(b) (currently up and running) was built according to the five following situations. It was first enhanced to also address the picasa service (b_1, b_2), provided by Google. Second, we also added a cache mechanism in front of the service calls (c_1, c_2, c_3). To limit the size of the retrieved set up to a given threshold, third we introduced a truncation mechanism (d_1). Then, we introduced a shuffling activity (e_1) to obtain different pictures each time. Finally, we introduced a timer mechanism to log the execution time of the shuffle activity (f_1, f_2).

Challenges. As we saw, even on a simple example, the need for evolution to react to business changes in the context of a business process is important. Therefore we consider in this article evolution as the integration of a set of changes in a legacy process. These changes are expressed separately by different stakeholders, and often introduce activities to be executed concurrently with the legacy process (*e.g.*, invoking picasa should not impact the invocation of flickr). Thus, the mechanism used to support these evolutions should not introduce unwanted waiting between activities. More critical interferences such as non-determinism can also be introduced in legacy processes. The manual detection of such interferences is difficult in front of thousands of activities and relations. Moreover, allowing a human to edit manually a business process does not ensure strong properties (*e.g.*, preservation of the pre-existing behavior) on the obtained process. The key idea of this contribution is to provide a formal meta-model supporting such evolution in an automatic way, *i.e.*, composing unforeseen changes with legacy processes and supporting interferences detection.

3. ADORE: modeling business process with first-order logic

We define Adore to address these challenges. The Adore activity meta-model is directly inspired by the Bpel language grammar expressiveness. The Bpel is defined as *"a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners"* [3]. The Bpel defines nine different kinds of atomic activities (*e.g.*, service invocation, message reception and response) and seven composite activities (*e.g.*, sequence, flow, if/then/else), plus additional mechanisms such as transactions and message persistence. We decide to extract from the Bpel all concepts necessary *"to describe the behavior of a business process"*. From the activity expressiveness point of view, Adore can be seen as a simplification of the Bpel concepts. We focus here on the definition of an activity kernel that support the definition of business processes in the large, without introducing a strong dependency with Bpel-specific concepts. Thus, we do not consider in Adore such notions (*e.g.*, correlation set). A coarse-grained description of the Adore meta-model is depicted in Fig. 2 using the class-diagram formalism.

In Adore, a business process is defined as a partially ordered set of activities. It defines several variables, used as inputs or outputs by its activities. Binary relations are used to schedule the activity set. A process can model an orchestration

[1] http://www.jseduite.org.