Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

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

Supporting the evolution of event-driven service-oriented architectures using change patterns

Simon Tragatschnig, Srdjan Stevanetic*, Uwe Zdun

Software Architecture Research Group, Faculty of Computer Science, University of Vienna, Vienna, Austria

A B S T R A C T

Context: The components of an event-driven service-oriented architecture (EDSOA) are composed in a highly decoupled way, facilitating high flexibility, scalability and concurrency in SOA systems. Evolving an EDSOA is challenging because the absence of explicit dependencies among constituent components makes understanding and analysing the overall system composition difficult. The evolution of EDSOAs typically happens by performing a series of primitive changes—which can be described formally as change primitives.
Objective: In this article, we present our change pattern based approach for managing the EDSOA evolution as a novel design method supporting EDSOA evolution. The change patterns operate on a higher abstraction level than change primitives.
Method: To evaluate our approach, we have compared both time and correctness of changes in a controlled experiment comparing the understanding and performing of changes in EDSOAs. The experiment has been conducted with 90 students of the Software Architecture course at the University of Vienna. We compare the efficiency of 3 sets of change operations for modifying a given system architecture to obtain a desired architecture: a minimal set of 3 change patterns, an extended set of 5 change patterns, and a minimal set of 4 change primitives.
Results: Our results show that change patterns based evolution requires significantly less time to capture a similar level of correctness as the evolution based on change primitives, presuming that a certain level of transformation complexity is required. Furthermore, we did not observe a significant difference in the correctness level nor in the time required to perform the changes using an extended pattern set compared to a minimal set of patterns.
Conclusions: We clearly show the feasibility of our approach by developing a design method and tool support using a model-driven tool chain consisting of 3 domain-specific languages and empirically evaluating the approach in a controlled experiment.

1. Introduction

In recent years, distributed event-driven architectures have become widespread in their use in several domains such as real-time control systems, stock market and fast trading, network intrusion detection, sensor networks, healthcare monitoring, mobile and wearable computing (see e.g. [20,26]). A main reason is that event-driven architectures provide solutions for developing distributed systems that facilitate high scalability, flexibility, and concurrency [20]. An event-driven architecture typically comprises a number of computational or data handling elements (i.e., components, actors) that communicate with each other by sending and receiving events [20]. Each component may independently perform a particular task, for instance, access a data storage, dispense cash from a credit card, or interact with users. Nowadays the components or actors in event-driven architectures are typically services, leading to a combination of event-driven and service-oriented architecture concepts, coined with the term event-driven service-oriented architecture (EDSOA) [13,21]. More precisely, we use the term to describe event-driven architectures that are used to realize flexible service communication and orchestrations [13,23,50].

The communication style used in EDSOAs is based on implicit invocations performed by publishing an event (or message) to an event channel (also called event bus or message broker) instead of explicit invocations where one component is directly called via a reference [26]. As the exchange of events among the components is performed through the event channel, every component is in principle unaware of the others. This way a high degree of flexibility in the system is supported.

* Corresponding author.
E-mail addresses: simon.tragatschnig@univie.ac.at (S. Tragatschnig), srdjan.stevanetic@univie.ac.at (S. Stevanetic), uwe.zdun@univie.ac.at (U. Zdun).

https://doi.org/10.1016/j.infsof.2018.04.005
Received 9 November 2016; Received in revised form 31 March 2018; Accepted 13 April 2018
0950-5849/ © 2018 Published by Elsevier B.V.

For example, the execution order of components can be changed (e.g., re-routing or adding some components) or any component can be replaced (e.g., with a bug-fixed or upgraded version) whilst the system is running. Additionally, EDSOAs support high scalability since the loosely coupled components can be executed concurrently and easily placed on different hardware nodes or virtual machines. However, the additional wanted degrees of flexibility, scalability, and concurrency through loose coupling might also increase the difficulty and uncertainty in understanding, maintaining and evolving these systems [8].

As requirements on software systems evolve over time, they have to be constantly maintained and changed [18]. More than one quarter of coding time is spent on implementing changes and investigating their impact [16]. By analyzing evolution of software systems, Weber et al. identify a set of change patterns that recur in many of existing software systems [45]. These patterns are specific for process-aware information systems (PAIS) where the execution of the software system is bound to a process schema, a prescribed rigid description of the behavior flow, and therefore, mostly cannot be changed during runtime or just slightly deviated from the initial schema [30,34,42]. As a result, these approaches are not easily applicable for EDSOAs where components are highly decoupled and the dependencies between components are subject to change at any time, even during the execution of the systems. Nevertheless, these patterns provide a basis for describing changes of the behavior in any information system.

To deal with the complexity and the large degree of flexibility of EDSOAs, a set of change operations that enable modifications in the system at different abstraction levels is proposed in our approach. Those change operations include low-level primitives (change primitives) for encapsulating the primitive change actions, such as adding or removing an actor or event, and high-level patterns (change patterns), that encompass a number of change primitives. An example of a change pattern is replacing an actor that represents a service call. This replacement pattern encompasses ‘removing and adding an actor’ primitives as well as ‘removing and adding events’ primitives. The provided set of patterns significantly extend the change patterns that are frequently occurring in most information systems [45] in order to deal with the specifics of EDSOAs. Hence our first research question addressed in this article is:

Research Question 1 (RQ1): Can the concept of change patterns (as defined for information systems by Weber et al. [45]) be used as a foundation for a design method for the evolution of EDSOAs, reflecting the specific changes required in EDSOAs?

In this article, we also investigate how the previously mentioned change operations (change patterns and change primitives) influence the process of performing changes or evolving an EDSOA. More precisely, we hypothesize a positive effect on the efficiency of performing changes in an EDSOA model using the change patterns compared to change primitive based changes. Hence, the second research question we studied was:

Research Question 2 (RQ2): If RQ1 can be answered positively, is there a significant positive influence on the efficiency of performing changes in an EDSOA model using the change patterns compared to change primitive based changes?

To study this research question, we conducted a controlled experiment where we compared the efficiency of change patterns and change primitives for modifying a given system architecture in order to obtain a desired one. The participants of our study were 90 students of the Software Architecture lecture at the University of Vienna. They were divided into 3 groups, and each of them was asked to modify a given system architecture (called the source architecture) using a given set of change operations in order to obtain a desired architecture (called the target architecture). For all groups the same 4 source/target pairs of EDSOA architecture models were given. The first group was provided with a set of 3 change patterns which also represents a minimal set of change patterns needed to perform any change in the system. The second group was provided with two additional patterns, to enable us to study to which extent additional higher-level abstractions help. Finally, the third group was provided with a set of 4 change primitives, which also represents a minimal set of primitives required to perform any change in the system. Our results show that, for the most complex model studied and the case where all models are considered together (the results for all studied models are summed up), the group with change primitives required significantly more time to reach a similar correctness level of pursued changes compared to the groups with change patterns. The obtained results provide empirical evidence that change patterns based evolution is generally more efficient than change primitives based evolution of EDSOAs, presuming that a certain level of transformation complexity is required. Moreover, the subjects used two additional patterns in the extended pattern set only to a limited extent and had problems with their correct application. No significant difference in the correctness level of pursued changes nor in the time required to capture those changes using the extended pattern set compared to the minimal set of change patterns was observed.

The major contribution of this article is the empirical study on efficiency of performing changes in EDSOAs, to find answers to our research questions RQ1 and RQ2.

In our previous work [1,37–39], we investigate and adapt change patterns in the context of event-based architectures dealing with the lack of prescribed execution descriptions and the potentialy aribrarily changed relationships between constituent elements of a system. In order to deal with the complexity and the large degree of flexibility of event-based architectures, in this article we combine our prior works into a novel design method for the evolution of EDSOAs. As another novel major contribution, we present an empirical study in which we evaluated our approach with 90 participants.

This article is organized as follows: In Section 2, we discuss the related work, and in Section 3, we describe the required background on EDSOAs and change operations in their context. We then describe our change pattern Based design method for supporting EDSOA evolution in Section 4. Next, in Section 5 we discuss our empirical study on the efficiency of performing changes in EDSOAs using our pattern based approach. Finally in Section 6, we summarize our main contributions.

2. Related work

2.1. Related works on change patterns

Starting from the seminal work on the evolution of software systems by Lehman [17], several techniques for supporting different types of system evolutions have been investigated in different application domains [2]. One of the important works presented by Weber et al. [31,44] identified a large set of change patterns that are frequently occurring in the most of today’s process-aware information systems (PAIS). The change patterns observed by Weber et al. targeted PAISs in which the execution order of the elements are prescribed at design time and unchanged or slightly deviated from the prescribed descriptions at runtime, therefore, not readily applicable for event-based systems where components are highly decoupled from each other. However, these patterns can be used as a basis for defining the corresponding change patterns for event-based systems (see e.g. [37–39]), since the structure of PAISs (i.e. process instances and their connections) can in principle be mapped to the structure of EDSOAs (i.e. event-based components and their connections). But, the specificities of EDSOAs in terms of e.g. the events that the EDSOA components send and/or receive or the execution domains need to be taken into account additionally. For instance, in PAISs the change primitives only deal with adding or deleting a node or an edge, while in EDSOAs additional primitives like e.g. replace an event of the component’s input and/or output port or remove a set of events from the port (see [37]) need to be considered.

Because of the loose coupled nature of EDSOAs, different variants of change patterns known from PAIS with different semantics may exist.