Science of Computer Programming 110 (2015) 78–103

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reasoning about software reconfigurations: The behavioural and structural perspectives

Nuno Oliveira, Luís S. Barbosa

HASLab, INESC TEC, Universidade do Minho, Braga, Portugal

ARTICLE INFO

Article history:
Received 8 July 2013
Received in revised form 24 May 2015
Accepted 29 May 2015
Available online 26 June 2015

Keywords:
Software reconfiguration
Software architecture
Coordination
Reo

ABSTRACT

Software connectors encapsulate interaction patterns between services in complex, distributed service-oriented applications. Such patterns encode the interconnection between the architectural elements in a system, which is not necessarily fixed, but often evolves dynamically. This may happen in response to faults, degrading levels of QoS, new enforced requirements or the re-assessment of contextual conditions. To be able to characterise and reason about such changes became a major issue in the project of trustworthy software.
This paper discusses what reconfiguration means within coordination-based models of software design. In these models computation and interaction are kept separate: components and services interact anonymously through specific connectors encoding the coordination protocols. In such a setting, of which Reo is a paradigmatic illustration, the paper introduces a model for connector reconfigurations, from both a *structural* and a *behavioural* perspective.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Complex distributed service-oriented systems require reliable and yet flexible architectures. A clear separation between typical *loci* of computation (*e.g.,* services or components) and the protocols that manage their interaction is at the heart of what are called *exogenous coordination* models [1]. Actually, interaction is mediated by software connectors, such as in Reo [2], which offer powerful "glue-code" to express such interaction protocols, while maintaining the envisaged separation of concerns.

These systems often evolve at runtime, entailing the need for dynamic reconfiguration of their interaction protocols. This is typically motivated by runtime faults, the need to cope with new requirements or the change in contextual conditions, which often degrades the expected levels of quality of service (QoS) to unacceptable levels [3].

Conventionally, an architectural reconfiguration mainly targets the manipulation (*e.g.,* substitution, update or removal) of components, often disregarding the connectors, or otherwise taking them as yet another component [4–8]. In exogenous systems, however, reconfigurations may target the interaction protocols themselves, *i.e.,* the connector's structure, as discussed, for instance, by C. Krause [9]. Reconfigurations may thus substitute, add or remove communication channels, or move communication interfaces between components, in order to restructuring a complex interaction policy.

Connector reconfiguration mechanisms play, in this setting, a major role to express change and adaptation of interaction protocols. Moreover, identifying and understanding the consequences of applying these reconfigurations is an important issue for the correct design of reconfigurable systems.

E-mail address: nunooliveira@di.uminho.pt (N. Oliveira).

http://dx.doi.org/10.1016/j.scico.2015.05.013
0167-6423/© 2015 Elsevier B.V. All rights reserved.

This paper combines and extends our previous work [10,11], to set up a conceptual framework for modelling and reasoning about reconfigurations of software connectors. Connectors are syntactically represented as graphs of communication primitives (*e.g.,* channels), referred to as *coordination patterns*, whose nodes stand for interaction points. Edges are labelled with identifiers and types which characterise their behaviour. The framework defines a number of elementary reconfiguration primitives, as well as how they can be combined to yield 'big-step' reconfiguration patterns able to transform significant parts of an architecture. Two complementary perspectives are introduced to reason about coordination pattern reconfigurations. One is *structural* and independent of whatever semantics is chosen for coordination patterns. It is concerned with requirements that, for example, enforce that during a reconfiguration a specific type of communication primitive remains attached to another specific primitive or a set of primitives. Such properties are specified in a propositional hybrid logic interpreted over the graph underlying the coordination pattern. The second perspective, on the other hand, is *behavioural* and relies on the specific semantics of coordination patterns. It may be of use, for example, to discuss to what extent a reconfiguration preserves the original interaction *behaviour* of a configuration.

It is important to emphasise, however, that reconfigurations in this paper are regarded from a static point of view. Intended for the software design phase, this approach allows the software architect (*i*) to specify connector reconfigurations; and (*ii*) to express and analyse their properties, either from a behavioural or structural perspective. However, the problematic of their dynamic application, including possible mechanisms for keeping the system's consistency along a run-time reconfiguration, is not addressed here.

Contributions The main contributions of this paper are (*i*) the development of a model for connector reconfigurations, their composition and application; (*ii*) a formal framework for comparing reconfigurations along the behavioural and the structural dimensions; and (*iii*) a case study from the e-healthcare domain illustrating the approach. Our previous work reported in references [10] and [11] introduced a notion of a reconfiguration pattern and discussed a first experiment on the use of a hybrid logic to express structural properties of coordination protocols, respectively. Both topics, however, are largely developed in this paper, leading to a new semantic model and a number of results for the logic, including a characterisation of bisimilarity and the proof of a Hennessy–Milner-like theorem on the equivalence between the assertion of two models being bisimilar and satisfying the same hybrid formulas.

Outline Coordination patterns are discussed in the next section paving the way to the detailed characterisation of reconfiguration operations in Section 3. The latter are combined to yield 'big-step' reconfiguration patterns. The following two sections introduce mechanisms for reasoning about connector reconfigurations from two orthogonal perspectives: a *behavioural* one (in Section 4), relying on whatever semantics is chosen for the underlying coordination model, and a *structural* one (in Section 5), in which properties of channel interconnection are expressed in a variant of propositional hybrid logic. Moreover, Section 5 defines bisimilarity for the structural models and proves a Hennessy–Milner-like theorem. A case study, taken from the e-healthcare domain, is discussed in Section 6 to illustrate the application of the approach proposed in the paper. Related work is discussed in Section 7. Finally, Section 8 concludes the paper and points out a number of open issues.

Notation Standard mathematical notation, namely for logic, is used throughout the paper. Maybe not so common is the representation of the powerset of a set X by 2^X and its extension to functions: $2^f(X) = \{f\, x \mid x \in X\}$. Function the and projections π_i, for i a natural number, are used to retrieve the unique element of a singleton set and the ith component of a tuple, respectively. Finally, if S is a set of sets notation $\bigcup S$ refers to the (iterated) union of all its elements, *i.e.,* $\bigcup S = \{x \in X \mid X \in S\}$.

2. Coordination patterns

Software connectors encoding reusable solutions for architectural problems in distributed, loosely-coupled systems are called in this paper *coordination patterns*. They are specified as graphs whose nodes represent interaction points and edges, standing for *communication primitives*, are labelled by pairs composed of their identifier and type. To keep exposition concrete, the Reo coordination model [1,2] (in particular, its set of primitive channels) is adopted, in the sequel, to provide types (and consequently a semantics) to the communication primitives in a coordination pattern.

In order to keep the paper reasonably self-contained, a brief introduction to Reo is given below. Coordination patterns are introduced afterwards.

2.1. A primer on Reo

Reo [1,2] is a popular model for exogenous service coordination based on channels. It is compositional, in the sense that complex coordination structures are obtained from the combination of channels.

2.1.1. Channels, nodes and connectors

A Reo channel is a point-to-point communication device with exactly two directed ends and a behaviour (or coordination protocol) defined within a specific semantic model. A channel *end* may accept or dispense data, in which cases it is said to be a *source*, or a *sink* end, respectively. Normally, a channel has one source and one sink end, but Reo also allows channels to

Download English Version:

https://daneshyari.com/en/article/433209

Download Persian Version:

https://daneshyari.com/article/433209

Daneshyari.com