Science of Computer Programming 127 (2016) 2–23

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

On-the-fly construction of provably correct service compositions – templates and proofs [☆]

Sven Walther [*], Heike Wehrheim

Department of Computer Science, Paderborn University, Germany

A R T I C L E   I N F O

Article history:
Received 11 February 2015
Received in revised form 1 March 2016
Accepted 5 April 2016
Available online 19 April 2016

Keywords:
Verification
Hoare-calculus
Templates
Service compositions
Correctness by construction

A B S T R A C T

Today, service compositions often need to be assembled or changed on-the-fly, which leaves only little time for quality assurance. Moreover, quality assurance is complicated by service providers only giving information on their services in terms of *domain specific* concepts with only limited semantic meaning.

In this paper, we propose a method for constructing service compositions based on *pre-verified templates*. Templates, given as workflow descriptions, are typed over a (domain-independent) template ontology defining concepts and predicates. Their meaning is defined by an *abstract semantics*, leaving the specific meaning of ontology concepts open, however, only up to given *ontology rules*. Templates are proven correct using a Hoare-style proof calculus, extended by a specific rule for service calls. Construction of service compositions amounts to instantiation of templates with domain-specific services. Correctness of an instantiation can then simply be checked by verifying that the domain ontology (a) adheres to the rules of the template ontology, and (b) fulfills the constraints of the employed template.

1. Introduction

Concepts like component-based software engineering (CBSE) or service-oriented architectures (SOA) ease the construction of software by combining off-the-shelf components or services to compositions. Today, such compositions often need to be assembled or changed *on-the-fly*, thereby imposing strong timing constraints on quality assurance, resulting from the need of fully automated verification and validation systems. "Quality" of service compositions might refer to either non-functional properties (like performance [1]), or functional requirements like adherence to protocols (e.g., [2]), to given pre- and postconditions [3], or to properties specified with temporal logic [4]. Quality assurance methods typically translate the composition (e.g., an architecture model, or a workflow description) into an analysis model, which captures the semantics of the composition. At the best, the quality analysis can be carried out automatically.

Both the transformation into the analysis model and the analysis itself are time-costly and thus difficult to apply in on-the-fly composition scenarios. Additionally, depending on the analysis model, *automatic* analysis it not necessarily possible (e.g., when using theorem provers).

[☆] This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

[*] Corresponding author.
E-mail addresses: swalther@mail.upb.de (S. Walther), wehrheim@upb.de (H. Wehrheim).

http://dx.doi.org/10.1016/j.scico.2016.04.002
0167-6423/© 2016 Elsevier B.V. All rights reserved.

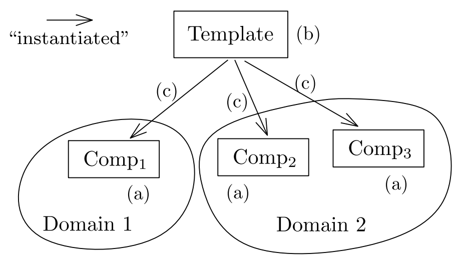

Fig. 1. Standard scenario: Full analysis for every composition (a). Our approach: Full analysis for templates (b), then check side conditions during instantiation (c).

In this paper, we propose a technique for service composition and analysis based on *templates*. Templates can capture known compositional patterns, and thus allow for the generally proven principle of pattern usage in software engineering [5]. In this paper, templates are workflow descriptions with *service placeholders*, which are replaced by concrete services during instantiation. If a template is shown to be correct, then all of its (valid) instantiations will be *correct by construction*. Every template specification contains pre- and postconditions (with associated meaning "if precondition fulfilled then postcondition guaranteed"), and a correct template provably adheres to this specification. To verify correctness of templates, we provide a Hoare-style proof calculus as a means to prove their correctness. Fig. 1 gives an overview on the motivating scenario.

The definition of "correctness" as well as giving a proof calculus for templates, however, poses a non-trivial task on verification. Since templates should be usable in a wide range of contexts and the instantiations of service placeholders are unknown at template design time, we cannot give a fixed semantics to templates. Rather, the template semantics needs to be *parameterized* in usage context and service instantiation. A template is only correct if it is correct for all (allowed) usage contexts. Similarly, a useful proof calculus has to be applicable in all possible contexts and service instantiations. We guarantee this by defining a proof calculus which is *parameterized* in usage contexts and template-specific *constraints*.

Technically, we capture the usage contexts by *ontologies*, and the interpretation of concepts and predicates occurring therein by *logical structures*. A *template ontology* defines the concepts and predicates of a template. Furthermore, a template specification contains *constraints* defining additional conditions on instantiations. These constraints allow us to verify the correctness of the template despite unknown usage and unknown fixed semantics. A template instantiation replaces the template ontology with a homomorphous *domain ontology*, and the service placeholders with concrete services of this domain. Verification of the instantiation then amounts to checking whether the (instantiated) template constraints are valid within the domain ontology, and thus can be carried out on-the-fly.

Section 2 describes ontologies and logical structures. Section 3 continues with the syntax of templates, and Section 4 proceeds with their semantics and correctness. Section 5 presents a proof calculus for templates, Section 6 an approach to utilize SMT solvers for proof automation. Section 7 explains instantiation and presents the central result of our approach: instantiation of correct templates yields correct service compositions, if constraints are respected. Section 8 discusses related work and an empirical approach to the same problem, and Section 9 concludes. This article is an extended version of [6], in addition giving a proof calculus for templates and proving soundness of this calculus with respect to the parameterized semantics of templates, as well as giving a translation to a SAT/SMT problem for automatic correctness checks.

2. Foundations

We assume service compositions to be assembled of services which are specified by a signature and pre- and postconditions. Languages to describe signatures with pre- and postconditions are already in use (e.g., OWL-S [3]). Such service descriptions usually rely on domain specific concepts. Services in use in a tourism domain might use concepts like restaurant, hotel and flight while those being employed in a medical domain use concepts like patient, medication and therapy. *Ontologies* are used to formally specify conceptualizations of domain knowledge [7]; their semantics can for instance be defined by *description logics* [8]. Ontologies can thus serve as a precise means of specifying our usage context of service compositions.

Simple notions of ontologies can be formalized using RDF (Ressource Description Framework, [9]), though typically the Web Ontology Language (OWL, [10,11]) is used. Basically, RDF allows for the specification of *triples*, relating two *concepts* with a *role*, or predicate. OWL is built on top of RDF, and comes with additional constructs for more high-level expressions like, e.g., transitivity of roles. Both RDF and OWL are W3C[1] recommendations.

There are complex relationships between concepts of an ontology which cannot be expressed by OWL. Therefore, *rule languages* of different expressivity are a topic of ontological research. A typical rule language is the Semantic Web Rule Language (SWRL, [12]), a W3C submission since 2004. It is built on top of OWL, and adds constructs to create implication-

[1] *World Wide Web Consortium*, http://www.w3.org.