On the (Compatible) Evolution of Services *

Vasilios Andrikopoulos

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart
Universitätsstraße 38
70569 Stuttgart
{firstname.lastname}@iaas.uni-stuttgart.de

Abstract: In an environment of constant change and variation driven by competition and innovation, a software service can rarely remain stable. Being able to manage and control the evolution of services is therefore an important goal for the Service-Oriented paradigm. This work extends existing and widely-adopted theories from software engineering, programming languages, service oriented computing and other related fields to provide the fundamental ingredients required to guarantee that spurious results and inconsistencies that may occur due to uncontrolled service changes are avoided. The presented work provides a unifying theoretical framework for controlling the evolution of services that deals with structural, behavioral and QoS level-induced service changes in a type-safe manner. The goal of the work is to ensure correct version transitions so that previous and future clients can use a service in a consistent manner.

The evolution of software due to changing requirements, technological shifts and corrective actions has been a well documented challenge for software engineering (and in particular for the field of Software Configuration Management) in the last decades [Leh96, ELvdH+05]. With regards to distributed and by extension service-oriented systems, however, a number of additional to traditional software system engineering challenges rise [BR00]. More specifically, large service networks consist of a number of services that potentially may belong to more than one organizations, making the identification and scoping of what constitutes the evolving system (so that maintenance activities can take place) non-trivial. More importantly, service-orientation is by definition based on a model of distributed ownership of services enabled by the loose coupling design principle, in the sense that services may be fully or partially composed out of third-party services that lay beyond the control of the service provider. In this context, the application of well-established techniques like refactoring or impact analysis becomes problematic at the very least.

Towards addressing these challenges, as part of our work in the context of EU’s Network of Excellence S-Cube¹, and culminating in [And10] and [ABP12], we have proposed a theoretical framework to manage the evolution of service interfaces in a type safe manner. The goal of this work is to assist service designers and developers in ensuring that changes to their services do not affect service consumers in a disruptive manner. For this purpose

*This work was partially funded by the FP7 EU-FET project 600792 Allow Ensembles.
we have adapted, reused and integrated some well established methods and techniques from both software engineering and computer science.

In particular with respect to [ABP12], the presented work starts with establishing a framework on which different compatibility definitions are positioned and connected with each other, distinguishing between two dimensions: horizontal/vertical (i.e. interoperability vs. substitutability) and backward/forward (i.e. provider- and consumer-oriented). A formal definition of compatibility is provided incorporating both identified dimensions based on type theory. For this purpose the meta-model for services first presented in [ABP08] is leveraged to facilitate type-based reasoning on service changes. The meta-model consists of elements and their relationships in three layers: structural (message-related), behavioral (w.r.t. the observable behavior of the services) and non-functional (QoS-related). The subtyping relation $\tau \leq \tau'$ is defined for the elements and relationships in each layer with semantics that depend on their types.

Building on these tools, the concept of \textit{T-shaped changes} is introduced as the set of change operations (add, delete, modify) $\Delta S$ that when applied to a service $S$ results in a fully compatible service description $S'$. Checking for (full) compatibility follows directly from definition and is realized as a short algorithm. Evaluation of the work focuses on two aspects: firstly showing how the proposed approach supersedes the established best practices for (Web) services evolution, and secondly demonstrating its efficacy and efficiency by means of a proof-of-concept realization, through which additional challenges were identified mainly w.r.t. the implementation technologies commonly used for Web services. Addressing a set of these challenges is the subject of ongoing work in the context of the Allow Ensembles EU project.

\section*{References}


