Challenges in a Service-Oriented World

by Wolfgang Reisig, Karsten Wolf, Jan Bretschneider, Kathrin Kaschner, Niels Lohmann, Peter Massuthe, and Christian Stahl

Interacting services raise a number of new software engineering challenges. To meet these challenges, the behaviour of the involved services must be considered. We present results regarding the behaviour of services in isolation, the interaction of services in choreographies, the exchangeability of a service, and the synthesis of desired partner services.

In a service-oriented world, enterprises use (Web) services to encapsulate parts of their process logic. The Web Service Business Process Execution Language (BPEL) is an established standard to describe Web services. Usually designed in isolation, a service must nevertheless properly interact with other services at run time. A number of techniques have been suggested to check the compatibility of service interfaces and to discover pairs of semantically matching services. The problem of checking behavioural compatibility, however, is rarely addressed. Figure 1 shows typical examples of behavioural incompatibility.

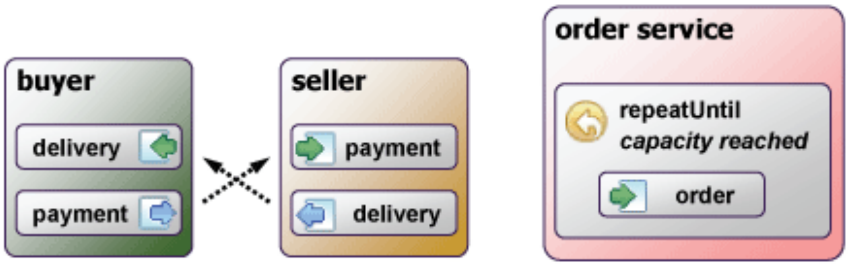

Figure 1: Services with incompatible behavior (the buyer and seller services wait for each other) and an ill-designed service (a partner of the order service would have to guess the services capacity).

In our joint research groups, and particularly in the Tools4BPEL project, we address behavioural compatibility and provide a series of techniques for detecting and repairing incompatibility, and for synthesizing behaviourally compatible partner services. Here we address the basics of this approach.

Behaviour of Services in Isolation
The designer of a service must guarantee that his/her service is controllable, ie there exists at least one properly interacting partner service. Controllability is thus a criterion that is as fundamental as the notion of soundness in the realm of business processes and workflows. Furthermore, a well-designed service should meet a number of additional properties such as executability of all activities. We suggest techniques for checking this kind of property based on the static analysis of BPEL code and model checking the state space of services.

Interaction of Services in Choreographies
Although a service might be well designed in isolation, it may cause a deadlock in a choreography of services. To prevent such deadlocks, designers must know either how other services in the choreography behave or how their particular services are supposed to behave. This means public views or operating guidelines can be used. A public view of a service describes communication from the point of view of the service itself. It is an abstracted version of the original service. An operating guideline of a service represents the possible communication behaviours of partners of the service. It is comparable to user instructions shipped with real-world devices. Both public views and operating guidelines support a number of scenarios, in particular service discovery. We propose algorithms for computing public views and operating guidelines.

Exchanging Services
A company may wish to exchange one of its services for another that is better suited for some purposes. The exchanged service should mimic all the interaction capabilities with existing partners. We differentiate between exchange at design time and exchange at run time. While exchange at design time concerns only new instances of a service, exchange at run time takes care of migrating instances of a service that are already running. As a particular scenario of exchangeability, we study contract-based compositions of services and develop notions of accordance between a contract and some local part of an actual implementation. If all local implementations accord with the contract, the overall composition is guaranteed to behave correctly.

Synthesis of Desired Partner Services
We develop algorithms that synthesize properly interacting communication partners of a given service. These algorithms also support the validation of a given service, test-case generation for a service, and the construction of adapters between services. Adapters can compensate for incompatibilities between services (see Figure 2).

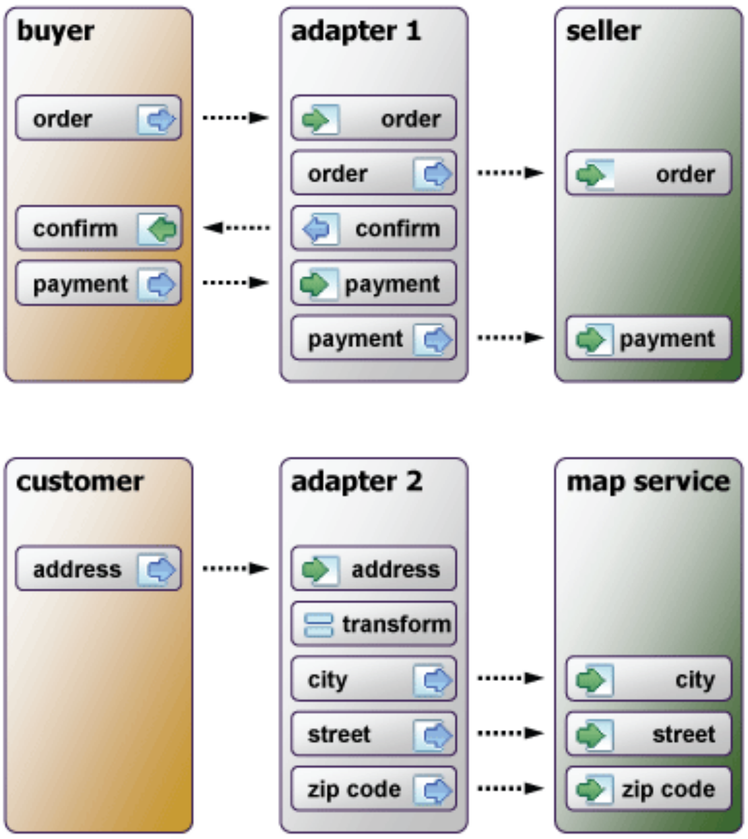

Figure 2: Adapters can compensate behaviorally incompatible services. Adapter 1 generates a confirmation message to resolve a deadlock. Adapter 2 transforms an address message into its constituents.

Diagnosis
It would be valuable for a designer to know why a service is not controllable. Such services typically exhibit static structures that can be conceived as responsible for incompatibilities. Knowing the reason helps the designer to remedy the problem. Therefore, we work on diagnosing which structures in a service are responsible for its misbehaviour and on their visualization.

Formal Semantics
In order to be independent of existing service description languages such as BPEL, we suggest a formal Petri-net-based modelling technique for services. The above-mentioned analysis techniques and algorithms are based on our modelling technique. This ensures their long-term applicability and potential use for various service description languages. All algorithms are implemented in our tool Fiona. This tool has a range of capabilities, including generation of operating guidelines and synthesis of communication partners. In addition, service models based on Petri nets are subject to our dedicated model checker LoLA.

We have chosen BPEL as one example of a Web service description language and implemented a compiler BPEL2oWFN, which translates back and forth between BPEL and our service model. Our approach hides the existence of the Petri net model from the users, depicting the analysis results in the BPEL code. Other (Web) service description languages can be analysed by the same techniques, assuming a corresponding compiler.

Links:
http://www.informatik.hu-berlin.de/top/
http://wwwteo.informatik.uni-rostock.de/ls_tpp/
http://www.informatik.hu-berlin.de/top/tools4bpel/

Please contact:
Wolfgang Reisig
Humboldt-Universität zu Berlin, Germany
Tel: +49 30 2093 3066
E-mail: reisig□informatik.hu-berlin.de

Karsten Wolf
Universität Rostock, Germany
Tel: +49 381 498 7670
E-mail: karsten.wolf□informatik.uni-rostock.de