School of Mathematics & Computer Science

Research Reports in

Computer Science

Proceedings of the Third European Young Researchers Workshop on Service-Oriented Computing

Monika Solanki
Barry Norton
Stephan Reiff-Marganiec

12-13 June 2008

Report No. CS-08-002

University of Leicester

Contents

A Relative Timed Semantics for BPMN

Peter Y. H. Wong and Jeremy Gibbons

Computing Laboratory, University of Oxford, United Kingdom
{peter.wong,jeremy.gibbons}@comlab.ox.ac.uk

Abstract. We describe a relative-timed semantic model for Business Process Modelling Notation (BPMN). We define the semantics in the language of Communicating Sequential Processes (CSP). This model augments our untimed model by introducing the notion of relative-time in the form of delays over non-deterministic ranges. By using CSP as the semantic domain, we show some properties relating the timed semantics and BPMN's untimed process semantics based on CSP's refinements. Our timed semantics allows behavioural properties of BPMN diagrams to be mechanically verified via automatic model-checking provided by the FDR tool.

1 Introduction

Modelling of business processes and workflows is an important area in software engineering. Business Process Modelling Notation (BPMN) allows developers to take a process-oriented approach to modelling of systems. In our previous work [11] we have given an untimed process semantics to a subset of BPMN in the language of CSP [9]. However due to the lack of notion of time, it is not possible to model precisely concurrent activities with timing information, this is particularly important when specifying business collaboration where the coordination of one business participant depends on the execution order of another participant's activities. Consider, for example, Figure 1 shows a trivial business collaboration between participants $p1$ and $p2$. Clearly we would like to know what temporal properties are required for $p1$ and $p2$ to be compatible in the collaboration. In the remaining sections we describe briefly our semantic model, show some properties relating the timed and untimed modes based on CSP refinements, and revisit the example to show how the relative-timed model may be used to verify compatibility between collaboration participants and how such a property can also be specified in BPMN. We conclude this paper with a comparison with related work. We assume readers have basic knowledge of the mathematical notations Z [14] and CSP [9]. The formal definition of the timed model may be found in the extended version of this paper [12].

2 Syntax and Timed Semantics of BPMN

States in our subset of BPMN can either be pools, tasks, subprocesses, multiple instances or control gateways; they are linked by sequence, exception or message

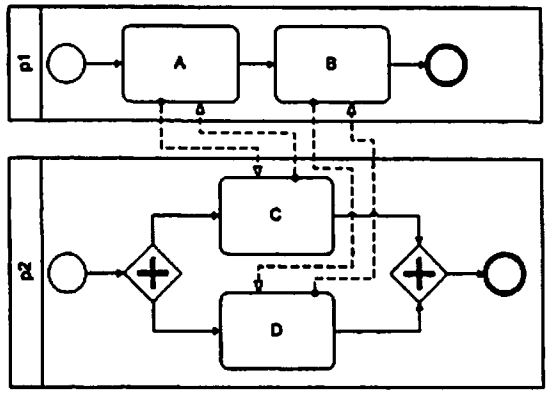


Fig. 1. A trivial business collaboration

flows; sequence flows can be either incoming to or outgoing from a state and have associated guards; an exception flow from a state represents an occurrence of error within the state. We assume each atomic task takes a finite positive amount of time to be executed. Message flows represent unidirectional communication between states of different pools where each pool forms a container for some business processes.

We use Z [14] to define the BPMN's abstract syntax and semantic function. In the remaining section we summarize our syntactic description and timed model of BPMN. Our timed model permits automatic translation, requiring no user interaction. According to the specification [7], each BPMN state type has associated attributes describing its properties and our syntactic definition has included some of these attributes. For example, we define each type of state with the free type *Type*, a partial definition of which is given by :

$$Type ::= agate \mid start \mid end\langle\langle \mathbb{N} \rangle\rangle \mid task\langle\langle Task \rangle\rangle \mid miseq\langle\langle Task \times \mathbb{N} \rangle\rangle$$

Note the number of loops of a sequence multiple instance state type is recorded by the integer in the constructor function *miseq*. We describe each state by recording its content type and its sets of transitions and message flows. Each atomic task state also specifies a delay range, $min \,..\, max$, of type *Range*, which it chooses non-deterministically, where *Time* records duration that conforms to W3C standards [15, Section 3.2.6].

$$Range \mathrel{\widehat=} [min, max : Time \mid min \leq_T max]$$

Each BPMN diagram encapsulated by a *pool* is a local diagram and represents an individual business participant in a collaboration, built up from a well-configured finite set of well-formed states [12]. While we associate each local diagram with a unique name, a global diagram, representing a business

collaboration, is built up from a finite set of names, each associate with its local diagram and we associate each global diagram with a unique name.

We define a timed semantic function which takes a syntactic description of a global diagram, describing a collaboration, and returns the CSP process that models the timed behaviour of that diagram. That is, the function takes one or more *pool* states, each encapsulating a local diagram representing an individual participant within a business collaboration and returns a parallel composition of processes each corresponding the timed behaviour of the individual participants. For each local diagram, the relative-timed semantics is the partially interleaving of two processes defined by an *enactment* and a *coordination* functions:

- The enactment function returns the parallel composition of processes, each corresponds to the untimed aspect of a state of the local diagram; this is essentially our untimed semantics of local diagrams [11].
- The coordination function returns a single process for coordinating that diagram's timed behaviour, it essentially implements a variant of *two-phase functioning approach* adopted by timed coordination languages like Linda [5].

The complete definition of the *enactment* and *coordination* functions may be found in our longer paper [12]. A prototype of the semantic function, implemented in the functional programming language Haskell, can be found in our web site[1].

3 Timed Analysis

The following are some results of the timed model. We say a diagram is *timed* if it contains timing information and *untimed* otherwise, every timed diagram is a *timed variant* of another untimed diagram, i.e. an untimed diagram augmented with timing information. Below is an intuitive property about timed variation.

Proposition 1. ***Untimed Invariance*** *For any untimed local diagram, there exists a (infinite) set of timed variant diagrams such that each of the diagram in the set is failures-equivalent under the untimed semantics.*

The CSP behaviour models traces ($\mathcal{T}$), stable failures ($\mathcal{F}$) and failures-divergences ($\mathcal{N}$) admit refinement orderings based upon reverse containment [9]. A behavioural specification R can be expressed by constructing the "least" - that is, the most non-deterministic - process satisfying it, called the characteristic process P_R. Any process Q that satisfies specification R has to refine P_R, denoted by $P_R \sqsubseteq Q$. One common behavioural specification any process would like to satisfy is deadlock freedom. A local diagram is deadlock free when all its possible execution either completes successfully or is interrupted by an abort state. We define the process DF to specify a deadlock freedom specification for local diagrams where events *fin.n* and *aborts.n* denote successful execution and

[1] `http://www.comlab.ox.ac.uk/peter.wong/observation/`

interruption respectively [12].

$$DF = (\sqcap i : \Sigma \setminus \{|fin, aborts|\} \bullet i \rightarrow DF) \\ \sqcap (\sqcap n : \mathbb{N} \bullet fin.n \rightarrow Skip) \sqcap (\sqcap n : \mathbb{N} \bullet aborts.n \rightarrow Stop)$$

Definition 1. *A local diagram is deadlock free iff the process, corresponding to the diagram's behaviour, failures-refines DF.*

Proposition 2. ***Deadlock Freedom Preservation*** *For any process P, modelling the behaviour of an untimed local diagram, such for its set of timed variant diagrams I, in which each diagram's behaviour is given by Q_i*

$$DF \sqsubseteq_{\mathcal{F}} P \Rightarrow \forall i : I \bullet DF \sqsubseteq_{\mathcal{F}} Q_i$$

Proof. (Sketch.) Due to the notion of CSP's responsiveness [8], we may proceed by showing for any local diagram, its timed coordination process is a *responsive plug-in* to its enactment process. Informally a process Q is a responsive plug-in to P if Q is prepared to cooperate with the pattern set out by P for their shared interface.

We say a behavioural property is time-independent if the following holds

Definition 2. ***Time Independence*** *A behavioural specification process S is time-independent with respect to some untimed local diagram, whose behaviour is given by process P iff for its set of timed variant diagrams I, in which each diagram's behaviour is given by Q_i such that*

$$S \sqsubseteq_{\mathcal{F}} P \Rightarrow \forall i : I \bullet S \sqsubseteq_{\mathcal{F}} Q_i$$

As a consequence of Proposition 2, traces refinement [9] and responsiveness [8], we can generalise timed-independent specifications by the following result.

Proposition 3. *A specification process S is time-independent with respect to some untimed local diagram, whose behaviour is given by the process P iff*

$$S \sqsubseteq_{\mathcal{F}} P \Leftrightarrow traces(S) \supseteq traces(P) \wedge deadlocks(S) \supseteq traces(P)$$

where $traces(P)$ is the set of possible traces of process P and $deadlocks(P)$ is the set of is the set of traces on which P can deadlock.

In the remaining section we revisit the example given in Figure 1. A more elaborate version of this can be found in the extended version of this paper [12]. This example shows a global diagram describing a collaboration between participants $p1$ and $p2$. While $p1$ performs task A then task B, $p2$ performs tasks C and D in an interleaving manner. We write $M1$ and $M2$ to denote the processes corresponding to the untimed behaviour of $p1$ and $p2$ respectively. Here we only show the definition of $M2$. First, we define $I2 = \{ start, as, c, d, aj, end \}$ to index the processes corresponding to the states in the participant $p2$. By applying the untimed semantic function upon the syntactic description of $p2$, we

obtain the process corresponding to it. Here the events $init.x$ denote incoming transitions of state x, $starts.t$ denote the initialisation of tasks t and $msg.a.b.m$ denote message flows from state a to b with message m. The set αP is the set of possible events performed by P.

$$\begin{aligned} M2' = \ & \text{let } C = (\Box\, e : (\alpha M2' \setminus \{fin.2\}) \bullet e \rightarrow C) \,\Box\, fin.2 \rightarrow Skip \\ & \text{in } (\,\|\, i : I2 \bullet \alpha P2(i) \circ P2(i))\, [\![\alpha M2']\!]\, C \end{aligned}$$

where for each i in $I2$, process $P2(i)$ defines the behaviour of state i.

$$\begin{aligned} P2(start) &= init.as \rightarrow fin.2 \rightarrow Skip \\ P2(as) &= (init.as \rightarrow (init.c \rightarrow Skip \,|||\, init.d \rightarrow Skip)\, ;\, P2(as)) \,\Box\, fin.2 \rightarrow Skip \\ P2(c) &= (init.c \rightarrow msg.a.c.mi \rightarrow starts.c \rightarrow msg.c.a.md \rightarrow init.aj1 \rightarrow P2(c)) \\ &\quad \Box\, fin.2 \rightarrow Skip \\ P2(d) &= (init.d \rightarrow msg.d.b.mi \rightarrow starts.d \rightarrow msg.b.d.md \rightarrow init.aj2 \rightarrow P2(d)) \\ &\quad \Box\, fin.2 \rightarrow Skip \\ P2(aj) &= ((init.aj1 \rightarrow Skip \,|||\, init.aj2 \rightarrow Skip)\, ;\, init.end \rightarrow P2(aj)) \,\Box\, fin.2 \rightarrow Skip \\ P2(end) &= init.end \rightarrow fin.2 \rightarrow Skip \end{aligned}$$

Now let's suppose tasks A, B and D have the delay range from 30 minutes to 1 hour and task C has the delay range from 45 minutes to 1 hour and 15 minutes. We define the timed behaviour of the collaboration by defining the process corresponding to individual participant's coordination using the coordination function [12]. For example Process $C3$ defines the coordination of participant $p2$.

$$\begin{aligned} C3 &= init.as \rightarrow (init.c \rightarrow Skip \,|||\, init.d \rightarrow Skip)\, ;\, C3_1 \\ C3_1 &= ((starts.d \rightarrow init.aj2 \rightarrow starts.c \rightarrow init.aj1 \rightarrow C3_3) \sqcap C3_2) \\ C3_2 &= (starts.c \rightarrow init.aj1 \rightarrow Skip \,|||\, starts.d \rightarrow init.aj2 \rightarrow Skip)\, ;\, C3_3 \\ C3_3 &= init.end \rightarrow fin.2 \rightarrow Skip \end{aligned}$$

and so we have process $T3$ defining the relative-timed semantics of participant $p2$, we write $T1$ to denote the process corresponding to $p1$ and it can be defined similarly using the semantic function. The timed semantics of their collaboration can hence be described by process TC.

$$\begin{aligned} T3 &= (M2' \,[\![\alpha M2 \cap \alpha C3]\!]\, C3) \setminus \{\!|init|\!\} \\ TC &= (T1 [\alpha M1 \,||\, \alpha M2]\, T3) \setminus \{\!|msg|\!\} \end{aligned}$$

CSP's refinement over $\mathcal{F}$ allows us to verifying the behaviour modelled by a BPMN diagram against another BPMN diagram, specifying the intended behaviour. We can describe such intended behaviour of the collaboration by defining a behavioural specification as the BPMN diagram $s1$ in Figure 2. We write $Spec$ to denote the process that models the untimed behaviour of $s1$, The refinement assertion $Spec \sqsubseteq_{\mathcal{F}} TC'$ checks whether the collaboration behaves as specified by the diagram $s1$. When we ask FDR to check this assertion the following counterexample in the form of a failure is given $(\langle starts.a \rangle, \Sigma)$. This tells us that after the collaboration deadlocks after participant $p1$ performed task A.

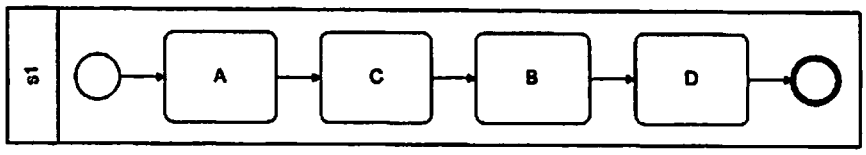


Fig. 2. A specification of the intended behaviour of collaboration between $p1$ and $p2$

A more detailed analysis reveals that after starting task A, participant $p1$ sent a message to $p2$'s task C. However, while task C's maximum delay is one minute and fifteen seconds, task D's maximum delay is only one minute. Since delay are chosen internally over a range without the coorperation of the environment, participant $p2$ can choose to perform task D before task C without any agreement with $p1$. In fact for $p1$ and $p2$ to be *compatible* with respect to the collaboration, the delay of task C cannot be longer than task D's.

4 Related Work and Conclusion

In this paper we introduced a relative-timed semantics for BPMN in CSP to model and reason about collaborations described in BPMN. We have adopted a variant of the two-phase functioning approach widely used in real-time systems and timed coordination languages [5]. We showed properties relating the untimed and timed models of BPMN for both local and global diagrams by using CSP's notion of responsiveness. We have also illustrated by an example how to use the timed model to verify compatibility between participants within a business collaboration. We have subsequently implemented a prototype of the semantic function in Haskell.

To the best of our knowledge, this paper describes the first relative-timed model for a collaborative graphical notation like BPMN. Some attempts have been made to provide timed models for similar notations such as UML activity diagrams [3, 4] and Workflow nets [6]. Both Guelifi et al. [3] and Eshuis [4] have defined their discrete timed semantic models in Clocked Transition System of which behavioural specifications are expressed as temporal logic formulae and verification are carried out via model checking; in Ling et al.'s work, they defined a formal semantics for a timed extension of van der Aalst's Workflow nets [10] in terms of timed Petri nets. Nevertheless, their semantics do not provide the level of abstraction required to model time explicitly in that they model discrete units of times which we believe may not be directly applicable to the business process developers whereas our definition captures the six-dimensional space defined by W3C standards [15, Section 3.2.6]. Also unlike BPMN, their target graphical notations and hence their semantic models are not designed for analyses of collaborations where more than one diagram is under consideration. Furthermore, our semantic model has been defined in correspondence to our earlier untimed

model [11] so that timed-independent behavioural properties may be preserved across both models.

As in the untimed settings, there exists many approaches in which new process calculi have been introduced to capture the notion of compatibility in collaborations and choreographies. Notable works include Carbone et al.'s End-Point and Glocal Calculi for formalising WS-CDL [2] and Bravetti et al.'s choreography calculus capturing the notion of choreography conformance [1]. Both these works tackled the problem of ill-formed choreographies, a class of choreographies of which correct projection is impossible. While the notion of ill-formed choreographies is similar to our definition of compatibility and the notion of contract refinement defined by Bravetti et al. [1] bears similarity to our definition of compatible class, they have defined their choreographies solely in terms of process calculi with no obvious graphical specification notation that could be more accessible to domain specialists.

Future work will include characterising the class of timed-independent behavioural properties suitable for BPMN; and applying the timed model to reason about empirical studies such as clinical trials against safety properties [13];

5 Acknowledgements

This work is supported by a grant from Microsoft Research. The authors are grateful to Bill Roscoe for his insightful advice on responsiveness during this work.

References

1. M. Bravetti and G. Zavattaro. Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In *Proc. of 6th International Symposium on Software Composition (SC'07)*, 2007.
2. M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown, and S. Ross-Talbot. A Theoretical Basis of Communication-Centred Concurrent Programming. Technical report, W3C, 2006.
3. N. Guelfi and A. Mammar. A Formal Semantics of Timed Activity Diagrams and its PROMELA Translation. In *APSEC05*, pages 283–290, 2005.
4. Hendrik Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling.* PhD thesis, University of Twente, 2002.
5. I. Linden, J.-M. Jacquet, K. D. Bosschere, and A. Brogi. On the expressiveness of timed coordination models. *Sci. Comput. Program.*, 61(2):152–187, 2006.
6. S. Ling and H. Schmidt. Time petri nets for workflow modelling and analysis. In *Proceedings of 2000 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3039–3044, 2000.
7. Object Management Group. *BPMN Specification*, Feb. 2006. `www.bpmn.org`.
8. J. N. Reed, J. E. Sinclair, and A. W. Roscoe. Responsiveness of interoperating components. *Form. Asp. Comput.*, 16(4):394–411, 2004.
9. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1998.

10. W. M. P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, 1997.
11. P. Y. H. Wong and J. Gibbons. A Process Semantics for BPMN, 2007. Submitted for publication.
12. P. Y. H. Wong and J. Gibbons. A Relative-Timed Semantics for BPMN, 2008. `http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmntime.pdf`.
13. P. Y. H. Wong and J. Gibbons. On Specifying and Visualising Long-Running Empirical Studies. In *Proceedings of International Conference on Model Transformation*, July 2008. To appear. Extended version available at `http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/transext.pdf`.
14. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.
15. XML Schema Part 2: Datatypes Second Edition, Oct. 2004. `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`.

Fixing Deadlocking Service Choreographies Using a Simulation-based Graph Edit Distance

Niels Lohmann

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Many work has been conducted to analyze services and service choreographies to assert manyfold correctness criteria. While errors can be *detected* automatically, the *correction* of defective services is usually done manually. This paper introduces a graph-based approach to calculate the minimal edit distance between a given defective service and synthesized correct services. This edit distance helps to automatically fix found errors while keeping the rest of the service untouched.

1 Introduction

In service-oriented computing [1], the correct interplay of distributed services is crucial to achieve a common goal. Choreographies [2] are a means to document and model the complex global interactions between services of different partners. BPEL4Chor [3] has been introduced to use BPEL [4] to describe and execute choreographies. Recently, a formal semantics for BPEL4Chor was introduced [5], offering tools and techniques to verify BPEL-based choreographies. Whereas it is already possible to automatically *check* choreographies for deadlocks or to synthesize participant services [6], no work was conducted in supporting the *fixing* of existing choreographies. This is especially crucial since fixing incorrect services is usually cheaper and takes less time than re-designing and implemeting a correct service from scratch. In addition, information on how to adjust an existing services can help the designer understand the error more easily compared to confronting him with a whole new synthesized service.

As a running example for this paper, consider the example choreography[1] of Fig. 1. It describes the interplay of a travel agency, a customer service, and an airline reservation system. The travel agency sends an offer to the client which either rejects it or books a trip. In the latter case, the travel agency orders a ticket at the airline service which sends either a confirmation or a refusal message to the customer. The customer service, however, does not receive the refusal message which leads to a deadlock in case the airline refuses the ticket order.

This design flaw can be easily corrected by adjusting the customer service. But even for this simple choreography, there is a variety of possibilities to fix the customer's service. Figure 2 depicts two possible corrections. Though both services would avoid the choreography to deadlock, the service in Fig. 2(a) is to be preferred over that in Fig. 2(b) as it is "more similar" to the original service.

[1] To ease the presentation, we use BPMN to visualize services and choreographies.

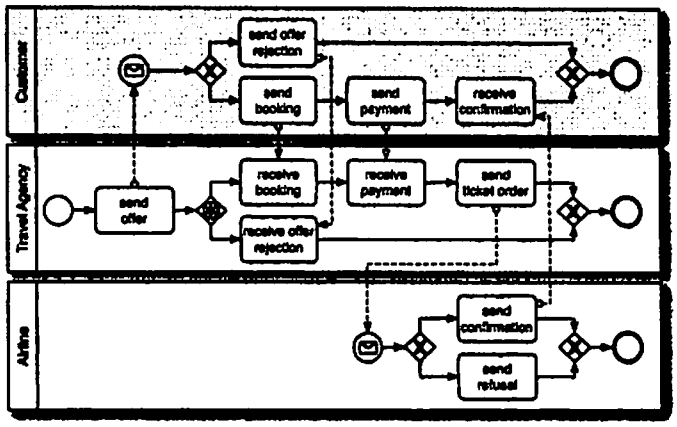
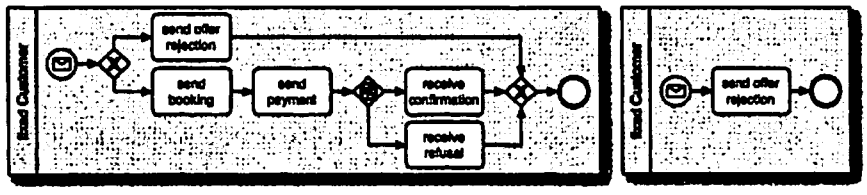


Fig. 1. Choreography between travel agency, airline, and customer. The choreography can deadlock, because the customer does not receive a refusal message from the airline.



(a) add receipt of refusal message (b) delete booking branch

Fig. 2. Two possible corrections of the customer service to avoid deadlocks.

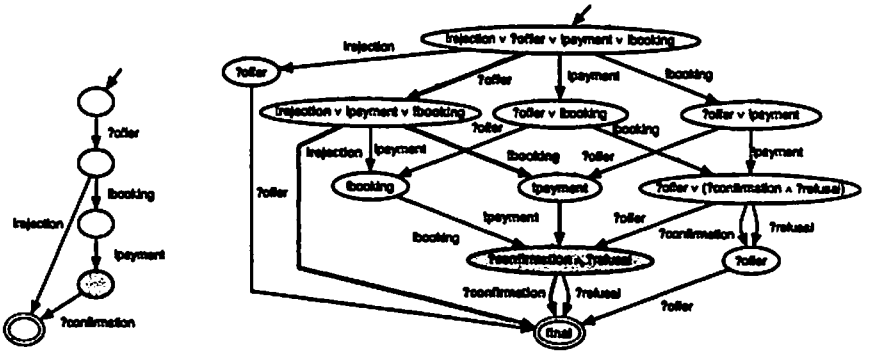
The goal of this paper (see [7] for a long version) is to formalize, systematize, and to some extent automatize the fixing of choreographies as above. We thereby combine existing work on characterizing all correctly interacting partners of a service with similarity measures and edit distances known in the field of graph correction. These approaches are recalled in Sect. 2. In Sect. 3, we define an edit distance that aims at finding the most similar service from the set of all fitting services. To support the modeler, we further derive the required edit actions needed to change the originally incorrect service. Finally, Sect. 5 is dedicated to a conclusion and gives directions for future research.

2 Background

2.1 Service Models and Partner Service Characterization

To formally analyze services, a sound mathematical model is needed. In the area of workflows and services, Petri nets are widely accepted formalism [8]. They combine a graphical notation with a variety of analysis methods. For real-life service description languages such as BPEL or BPEL4Chor exists a feature-complete Petri net semantics [9, 5]. To simplify the presentation, we abstract from

the structure of a service and complex aspects such as data or fault handling, and focus on the external behavior of services in this paper. To this end, we use automata to model the external behavior services.[2]



(a) automaton (b) operating guideline (OG)

Fig. 3. An automaton modeling the customer service of Fig. 1 (a) and the operating guideline of the composition of travel agency and airline service (b).

Figure 3(a) depicts an automaton modeling the external behavior of the customer service of Fig. 1. The edges are labeled with messages sent to (preceded with "!") or received from (preceded with "?") the environment. As services are usually not considered in isolation, their interplay has to be taken into account in verification. An important correctness criterion is *controllability* [10]. A service is controllable if there exists a partner service such that their composition (i. e., the choreography of the service and the partner) is free of deadlocks.

Controllability can be decided constructively: if a partner service exists, it can be automatically synthesized [10, 6]. Furthermore, there exists one canonic partner service that is most permissive; that is, it simulates any other partner service. The converse however, does not hold; not every simulated service is a correct partner service itself. To this end, the most permissive partner service can be annotated with Boolean formulae expressing which states and transitions can be omitted and which parts are mandatory. This annotated most permissive partner service is called an *operating guideline* (OG) [11].

Figure 3(b) depicts the OG of the composition of the travel agency and the airline. The disjunction of the OG's initial state means that a partner service must send a rejection, receive an offer, send a payment or send a booking in its initial state. Conjunctions in OG's states represent the fact that some choices cannot be influenced by the partner of the travel agency and the airline. Instead, the customer has to react on any decision of the travel agency and the airline. For example, the formula "?confirmation ∧ ?refusal" describes the fact that the

[2] Due to the close relationship between Petri nets and automata, there exist techniques to transform back and forth between the two formalisms.

airline decides whether to confirm *or* to refuse the booking and the traveler has to be ready to receive *both* resulting messages.

The automaton of Fig. 3(a) is simulated by the OG and fulfills all but one formula (satisfied literals are bold in Fig. 3(b)). It does not satisfy the formula "?confirmation ∧ ?refusal" of the OG's state (depicted gray) reached after sending the booking and payment, because the automaton does not receive a refusal message in this state. Beside the corrected services of Fig. 2, the OG characterizes 1,021 additional partner services.

2.2 Graph Similarities and Edit Distances

Graph similarities are widely used in many fields of computer science, for example for pattern recognition or in bio informatics. Cost-based distance measures adapt the *edit distance* known from string comparison [12] to compare labeled graphs (e. g., [13]). They aim at finding the minimal number of modifications (i. e., adding, deleting, and modifying nodes and edges) needed to achieve a graph isomorphism. These distance measures have the drawback that they are solely based on the *structure* of the graphs. Thus, they focus on the syntax of the graphs rather than their semantics. When a graph (e. g., an automaton) models the *behavior* of a system, similarity of graphs should focus on simulation of behavior rather than on a small edit distance (see Fig. 4).

Fig. 4. Automaton (a) and (b) simulate each other, but have a high edit distance. Automaton (b) and (c) have a low edit distance, but rather unsimilar behavior.

In [14], this problem is addressed and motivated by finding computer viruses by comparing a program with a library of control flow graphs. In that setting, classical simulation is too strict because two systems that are equal in all but one edge label *behave* very similar, but there exists no simulation relation between them. To this end, the authors introduce a *weighted weak quantitative simulation* function to compare states of two graphs. Whenever the two graphs cannot perform a transition with same labels, one graph performs a special stuttering step similar to τ-steps in stuttering bisimulation [15]. Stuttering is penalized using a cost function that assigns any pair of labels a cost. Minimizing the costs for stuttering then yields in maximal similarity.

3 Combining Edit Distance and Service Matching

The algorithm to calculate weighted weak quantitative simulation can be used as a similarity measure for automata or OGs, but has two drawbacks: Firstly, it is not an edit distance, as it does not give the modification actions needed to achieve

simulation. Secondly, it does not take formulae of the OG into account. Therefore, a high similarity between an automaton and a OG would not guarantee deadlock freedom as the example of Fig. 3 demonstrates.

3.1 Simulation-based Edit Distance

Given a state q_A of an automaton and q_O of an OG, the algorithm of [14] determines the best matching between the transitions of q_A and q_O. In addition, the automaton or the OG can stutter (i. e., remain in the same state). From these transition pairs that are used to calculated the similarity measure, we can derive the according edit actions (see Table 1).

Table 1. Deriving edit actions from transition pairs of [14].

automaton transition	OG transition	resulting edit action
a	a	keep transition a
a	b	modify transition a to b
a	*stutter*	delete transition a
stutter	a	insert transition a

3.2 Adding Formula-checking to the Edit Distance

As described earlier, a partner is represented by the OG if (i) the OG simulates the partner and (ii) the OG's formulae are fulfilled. The simulation-based edit distance does not respect the OG's formulae. Therefore, a partner created by this edit distance is not guaranteed to interact without deadlocks. Experiments showed that fixing these partner services by adding edges to fulfill the OG's annotations yield suboptimal results. To this end, we extend the algorithm of [14] not to statically take all outgoing transitions of an OG's state into account, but also check any formula-fulfilling subset of outgoing transitions. The algorithm now adds all edges necessary to fulfill the OG's formulae while trying to find the best matching assignment.

When comparing the gray states of Fig. 3, the "?confirmation" transitions match, but the OG's "?refusal" transition can only be matched to a stuttering step of the automaton. Thus, "insert transition ?refusal" would be the resulting edit action of the automaton's state. Figure 5 illustrates the result of the algorithm.

4 Complexity Considerations and Experimental Results

The original simulation algorithm of [14] to calculate a simulation between two service automata A_1 and A_2 needs to check $\mathcal{O}(|Q_{A_1}| \cdot |Q_{A_2}|)$ state pairs (Q_{A_i} is the set of states of A_i). The extension to respect the OG's formulae to calculate the edit distance between a service automaton A and an OG O takes the OG's formulae and the resulting label permutations (i. e., how the edges

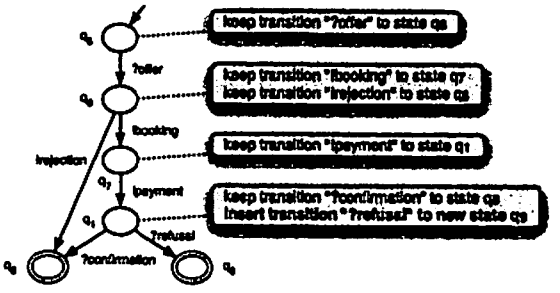


Fig. 5. Matching-based edit distance applied to the customer's service.

of a satisfying assignment are mapped to the edges of the service automaton) into consideration. The length of the OG's formulae is limited by the maximal degree of the nodes which again is limited by the interface I. Thus, for each state pair, at most $2^{|I|}$ satisfying assignments have to be considered. The number of permutations is again limited by the maximal node degree such that at most $|I|!$ permutations have to be considered for each state pair and assignment. This results in $\mathcal{O}(|Q_A| \cdot |Q_O| \cdot 2^{|I|} \cdot |I|!)$ comparisons.

Though the extension towards a formula-checking edit distance yields a high worst-case complexity, OGs of real-life services tend to have quite simple formulae, a rather small interface (compared to the number of states), and a low node degree. As a proof of concept, we implemented the edit distance in a prototype.[3] It takes an acyclic deterministic service automaton and an acyclic OG as input and calculates the edit actions necessary to achieve a matching with the OG. The prototype exploits the fact that a lot of subproblems overlap, and uses dynamic programming techniques [16] to cache and reuse intermediate results which significantly accelerates the runtime. We evaluated the prototype with models of some real-life services. In most cases, the edit distance could be calculated within few seconds. Table 2 summarizes the results.

The first seven servies are derived from BPEL processes of a German consulting company; the last four services are taken from the current BPEL specification [4]. Column "search space" of Table 2 lists the number of acyclic deterministic services characterized by the OG. All these services are correct partner services and have to be considered when finding the most similar service. The presented algorithm exploits the compact representation of the OG and allows to efficiently find the most similar service from more than 10^{2000} candidates.

For most services, the calculation only takes a few seconds.[4] The "Internal Order" and "Purchase Order" services are exceptions. The OGs of these services have long formulae with a large number of satisfying assignments (about ten times larger than those of the other services) yielding a significantly larger search space. Notwithstanding the larger calculation time, the service fixed by the calculated

[3] Available at `http://service-technology.org/rachel`.

[4] The experiments were conducted on a 2 GHz computer.

Table 2. Experimental results.

service	interface	states SA	states OG	search space	time (s)
Online Shop	16	222	153	10^{2033}	4
Supply Order	7	7	96	10^{733}	1
Customer Service	9	104	59	10^{108}	3
Internal Order	9	14	512	$> 10^{4932}$	195
Credit Preparation	5	63	32	10^{36}	2
Register Request	6	19	24	10^{25}	0
Car Rental	7	49	50	10^{144}	6
Order Process	8	27	44	10^{222}	0
Auction Service	6	13	395	10^{12}	0
Loan Approval	6	15	20	10^{17}	0
Purchase Order	10	137	168	$> 10^{4932}$	391

edit actions is correct by design, the difference between the erroneous and the fixed service is minimal, and the calculation time is surely an improvement compared to iterative manual correction.

5 Conclusion and Future Work

We presented an edit distance to find the necessary edit actions to correct a faulty automaton to interact without deadlock in a choreography. The edit distance (i.e., the actions needed to fix the automaton) can be automatically calculated using a prototypic implementation. Together with translations from and to [9, 17] BPEL processes and the generation of the OG [6, 11], a complete tool chain[5] to analyze and correct BPEL-based choreographies is available.

However, a lot of questions remain open. First of all, the choice *which* service to fix is not always obvious and needs further investigation. For instance, the choreography of Fig. 1 could also have been fixed by adjusting the airline service. In contrast, the service of the travel agency cannot be fixed with the presented approach: the composition of the customer and the airline service is not controllable and thus no OG exists. The diagnosis of uncontrollable services is subject of future work.

Another aspect to be considered in future research is the choice of the *cost functions* used in the algorithm, as it is possible to set different values for any transition pairs. Semantic information on message contents and relationships between messages can be incorporated to refine the correction. For example, the insertion of the receipt of a confirmation message can be penalized less than the insertion of sending an additional payment message. Beside the presented application of fixing choreographies, the edit distance might also be used in the context of service-oriented architectures (SOA): given a service requester as automaton and a service repository consisting of provider services' OGs, the

[5] See `http://service-technology.org/tools`.

edit distance might help to find the smallest adaptation of the requester to find a fitting provider. Finally, heuristics may help to increase the performance of the implementation by omitting suboptimal edit actions. For instance, guidance metrics such as used in the A* algorithm [18] may greatly improve runtime.

References

1. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Commun. ACM 44(4) (2001) 71–77
2. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. IJCIS 13(4) (2004) 337–368
3. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, IEEE (2007) 296–303
4. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (2007)
5. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 46–60
6. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: BPM 2006. Volume 4102 of LNCS., Springer (2006) 17–32
7. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, Springer (2008) to appear.
8. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1) (1998) 21–66
9. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 77–91
10. Schmidt, K.: Controllability of open workflow nets. In: EMISA 2005. Volume P-75 of LNI., GI (2005) 236–249
11. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. Volume 4546 of LNCS., Springer (2007) 321–341
12. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Dokl. 10(8) (1966) 707–710
13. Tsai, W., Fu, K.: Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans. on SMC 9(12) (1979) 757–768
14. Sokolsky, O., Kannan, S., Lee, I.: Simulation-based graph similarity. In: TACAS 2006. Volume 3920 of LNCS., Springer (2006) 426–440
15. Namjoshi, K.S.: A simple characterization of stuttering bisimulation. In: FSTTCS 1997. Volume 1346 of LNCS., Springer (1997) 284–296
16. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
17. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. Volume P-127 of LNI., GI (2008)
18. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans. Syst. Sci. and Cybernetics SSC-4(2) (1968) 100–107

Automatic Test Case Generation for Services

Kathrin Kaschner and Niels Lohmann

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
{kathrin.kaschner, niels.lohmann}@uni-rostock.de

Abstract. Service-oriented computing (SOC) proposes loosely coupled interacting services as building blocks for distributed applications. This distributed nature makes the guarantee of correctness a challenging task. Based on the specification of a service (the public view), we introduce an approach to automatically generate test cases for black-box testing to check for compliance between the specification and the implementation of a service whose internal behavior might be secret.

1 Introduction

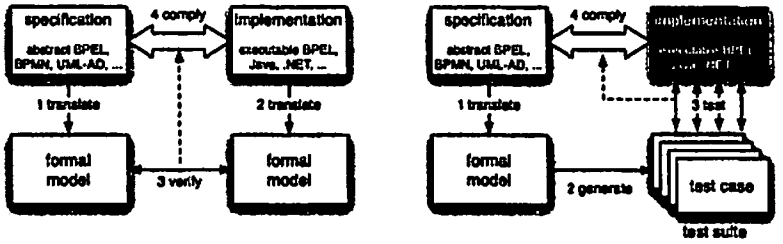
Software systems continuously grow in scale and functionality. Various applications of a variety of different areas interact and are increasingly dependent on each other. Then again, they have to be able to be adjusted to the rapidly changing market conditions. Accordingly complex is the development and changing of these distributed enterprise applications.

To face these new challenges a new paradigm has emerged: *service-oriented computing* (SOC) [1]. It uses *services* as fundamental elements for readily-creating, flexible and reusable applications within and across organizational boundaries. A service is an encapsulated, self-contained functionality. Usually, it is not executed in isolation, but interacts with other services through message exchange via a well-defined interface. Thus, a system is composed by a set of services.

A well known class of services are *Web services* which use WSDL to describe their interface and SOAP to exchange messages. To deploy a Web service, two steps have to be taken. Firstly, its publicly observable behavior including the partners of the service and the exchanged messages is described. This *specification* — sometimes called business protocol or public view — is then *implemented* in a second step. To this end, all necessary details about the binding, message buffers, instantiation, etc. are added. While usually the specification language (e. g., BPMN, UML activity diagrams, EPCs) differs from the implementation language (e. g., Java, .NET languages), BPEL can be used to both specify and implement Web services. The specification (called an abstract BPEL process) contains placeholders that are replaced by concrete activities in the implementation (the executable BPEL process). The implementation can thereby be seen as a refinement of the specification.

Obviously, the implementation should comply to its specification. In the field of Web services, this is crucial as the public views are used as documentation of how to interact with a service. Thereby, any correct interaction derived from

the public view should also be a correct interaction with the implementation. Compliance between an implementation and a specification can be shown by *verification*, cf. Fig 1(a). Thereby, the implementation and its specification have to be translated into a formal model (1, 2). Compliance between the models (3) then implies compliance between specification and implementation (4). Obviously, this approach can only be applied if the two models are available.



(a) verification (b) proposed black-box testing

Fig. 1. Approaches to check compliance between specification and implementation.

However, there are several scenarios in which a the formal model of the implementation is not available which makes verification impossible. For example, parts or the whole implementation might be subject to trade secrets. Furthermore, programming languages such as Java are — compared to high-level languages like BPEL — rather fine-grained and it is therefore excessively elaborate to create a formal model for such an implementation.

As an alternative, the implementation can be tested. The major disadvantage is that by *testing* only the presence of errors can be detected, but not their absence. To nevertheless be able to make a statement about the correctness of an implementation, a test suite with a significant number of test cases is necessary. Because only the interface of the implemented service is known and only the output of the service under test can be observed, this is called *black-box testing* in literature.

In this paper, we will present a novel approach to automatically generate all significant test cases to check compliance of the implementation and its specification, cf. Fig. 1(b). Therefore, we translate the specification into a formal model (1). This transformation is possible, because the specification is publicly available and usually not as complex as for example a Java program. The model of the specification is then the base for generating the test cases (2), with which we test the implementation (3). If a tests fails, we can conclude that the implementation does not comply to the specification (4).

The rest of the paper is organized as follows. In Sect. 2, we present a small example to motivate black-box Web service testing. The necessary formal models for services are recalled in Sect. 3. In Sect. 4, we sketch the test case generation approach. Section 5 is dedicated to related work. Section 6 concludes the paper and gives directions for future research.

2 Example

As an example for a service specification, consider the BPMN diagram in Fig. 2. It models the public view of a simple online shop of [2]. The shop internally decides after the user's login whether the user is a premium customer (upper branch) or a standard customer (lower branch). Standard customers have to accept the terms of payment before receiving an invoice.

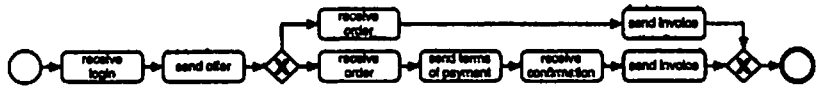
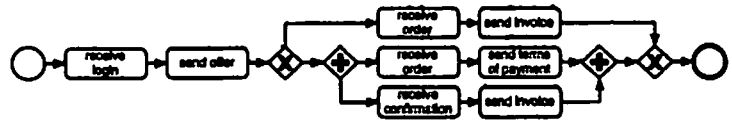
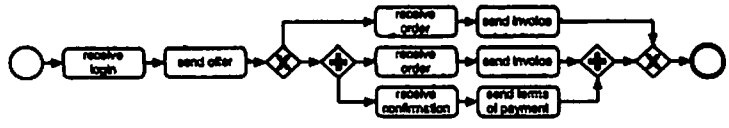


Fig. 2. A specification for a simple online shop.

To implement the online shop, a lot of details that have been left out in the specification have to be filled in. For example, the specification gives no information on how instantiation, message correlation, message queuing, fault handling, or logging is organized. Beside adding internal actions, also the order of communicating activities can be changed without jeopardizing compliance to the public view. After these modifications, it is not any more obvious that the resulting service complies to the original specification. Furthermore, the translation from the specification formalism (e. g., BPMN) into an implementation language (e. g., Java) is not always straightforward.



(a) correct implementation



(b) incorrect implementation

Fig. 3. Two implementations of the specification in Fig. 2.

Figure 3 shows two implementations of the online shop.[1] Implementation (a) executes the tasks for standard customers concurrently, yet still complies to the specification. The second implementation (b) further exchanges the invoice and the terms of payment message. This implementation does not comply to the specification as we will see later.

[1] To ease the presentation, we again use BPMN to diagram the implementations and focus on the message exchange between online shop and customer.

3 Modeling Services

As modeling formalism, we use *open workflow nets* (oWFNs) [3], a special class of Petri nets. They generalize classical workflow nets [4] by introducing an interface for asynchronous message passing. An important correctness criterion is *controllability* [5]. A service is controllable if there exists a partner service such that their composition (i.e., the choreography of the service and the partner) is free of deadlocks.

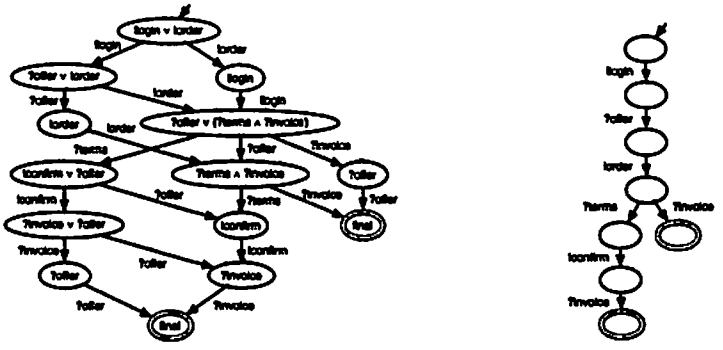
Controllability can be decided constructively: if a partner service exists, it can be automatically synthesized [2]. Furthermore, there exists one canonic *most permissive* partner which simulates any other partner service. The converse does, however, not hold; not every simulated service is a correct partner service itself. To this end, the most permissive partner service can be annotated with Boolean formulae expressing which states and transitions can be omitted and which parts are mandatory. This annotated most permissive partner service is called an *operating guideline* (OG) [6]. With translations back and forth between BPEL and oWFNs [7, 8], results of oWFNs can be directly applied to real-life Web services.

4 Generating Test Cases

To check whether an implementation of a Web service complies with its specification, we focus on the partner services of the service. That are services which interact without deadlock with the given Web service. The implementation is compliant to the specification if it does not exclude partner services: any service that communicates correctly with the specification (i. e., the public view) should also communicate correctly with the implementation. To this end, any partner service of the specification can be seen as test case. The complete test suite is then characterized by the OG of the model of the specification.

The OG of the specification of Fig. 2 is depicted in Fig. 4(a). It is an automaton whose edges are labeled with messages sent to (preceded with "!") or received from (preceded with "?") the environment. The annotations describe for each state which have to be present (see [6] for further details). In total, the OG characterizes 139 partners. However, it is possible to select a small subset of partners as test cases. On the one hand, the most permissive partner is a canonic test case which can be derived by omitting the formulae of the OG. This partner covers the maximal behavior of the specification. On the other hand, the formulae can be used to "split" the most permissive partner into several smaller test cases. These test cases then can be checked independently of each other, for example using several dedicated test servers. In addition, the set of test cases can be adjusted using behavioral constraints [9]. Constraints allow to only generate certain test cases, for instance only those that use a certain functionality or send messages in a given order.

One test case is depicted in Fig. 4(b). This test case interacts without deadlock with both the specification (Fig. 2) and the first implementation (Fig. 3(a)). The communication with the second implementation (Fig. 3(b)), however, deadlocks.



(a) OG of specification of Fig. 2 (b) a test case

Fig. 4. The OG (a) describes all 139 test cases. One of them (b) can be used to show that the implementation in Fig. 3(b) does not comply to the specification of Fig. 2.

If the customer is treated as standard customer, the online shop — after receiving the login and an order and sending the offer and the invoice — is left in a state where it awaits the confirmation message. This will, however, only be sent by the customer after receiving the terms of payment. The services wait for each other; the composition deadlocks. While this test case can be easily derived directly from the specification, manual test case generation becomes an increasingly tedious task for real-life specifications with thousands of partner services.

If the specification is given as abstract BPEL service, an existing tool chain[2] is directly applicable to translate the specification into an oWFN and to calculate the OG thereof (cf. [2]). Furthermore, the test cases can be automatically translated into BPEL processes [8]. The resulting test Web services can then be executed on a BPEL engine to test the implementation under consideration.

5 Related Work

Several works exist to systematize testing of Web services (see [10] for an overview). In [11, 12], white-box test architectures for BPEL are presented. The approaches are very similar to unit testing in classical programming languages, but not applicable in our black-box setting. Furthermore, the authors give no information about how to generate test cases.

There exist several software tools that support testing of Web services, mostly focusing on white-box testing. Some for example Oracle BPEL Process Manager [13] or Parasoft SOAtest [14] allow to automatically create test cases. However, neither of products support a systematic test case generation.

Test generation can be tackled using a variety of approaches such as control flow graphs [15], model checking [16], or XML schemas [17]. These approaches mainly focus on code coverage, but do not take the communicating nature of Web services into account. In particular, internal activity sequences are not necessarily enforceable by a test case. Therefore, it is not clear how derived test cases can

[2] See `http://www.service-technology.org/tools`.

be used for black box testing in which only the interface of the service under test is observable. Furthermore, none of the testing approaches takes the complex communication nature of Web services into account and mainly assume simple remote procedure calls (i. e., request-response operations).

6 Conclusion

We presented an approach to automatically generate test cases for services. These test cases are derived from the OG of a specification and can be used to test compliance of an implementation without explicit knowledge of the latter (black-box testing). The approach bases on oWFNs as formal model, and can be applied to any specification language to which a translation into oWFNs exists. To support other specification formalisms, a translation into oWFNs is necessary. Existing translations from BPMN or UML-AD into Petri nets can are likely to be adjustable to oWFNs easily.

In future work, we also want to consider negative test cases to test the absence of unintended partners. Furthermore, we plan to validate our test case generation in a BPEL test architecture (cf. [11]) using real-life case studies. Another important aspect is to add data (as far as specified) to the test case generation to refine the results.

References

[1] Papazoglou, M.P.: Agent-oriented technology in support of e-business. Communications of the ACM 44(4) (2001) 71–77

[2] Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: BPM 2006. Volume 4102 of LNCS., Springer (2006) 17–32

[3] Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3) (2005) 35–43

[4] Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1) (1998) 21–66

[5] Schmidt, K.: Controllability of open workflow nets. In: EMISA 2005. Volume P-75 of LNI., GI (2005) 236–249

[6] Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. Volume 4546 of LNCS., Springer (2007) 321–341

[7] Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 77–91

[8] Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. Volume P-127 of LNI., GI (2008)

[9] Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. Volume 4714 of LNCS., Springer (2007) 271–287

[10] Baresi, L., Nitto, E.D., eds.: Test and Analysis of Web Services. Springer (2007)

[11] Li, Z., Sun, W., Jiang, Z.B., Zhang, X.: BPEL4WS unit testing: Framework and implementation. In: ICWS 2005, IEEE (2005) 103–110

[12] Mayer, P., Lübke, D.: Towards a BPEL unit testing framework. In: TAV-WEB '06, ACM (2006) 33–42

[13] Oracle: BPEL Process Manager. (2008) http://www.oracle.com/technology/products/ias/bpel.
[14] Parasoft: SOAtest. (2008) http://www.parasoft.com.
[15] Yan, J., Li, Z., Yuan, Y., Sun, W., Zhang, J.: BPEL4WS unit testing: Test case generation using a concurrent path analysis approach. In: ISSRE 2006, IEEE (2006) 75–84
[16] García-Fanjul, J., Tuya, J., de la Riva, C.: Generating test cases specifications for BPEL compositions of Web services using SPIN. In: WS-MaTe 2006. (2006) 83–94
[17] Hanna, S., Munro, M.: An approach for specification-based test case generation for Web services. In: AICCSA, IEEE (2007) 16–23

The Need for a Choreography-aware Service Bus*

Oliver Kopp, Tammo van Lessen, and Jörg Nitzsche

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{oliver.kopp,tammo.van.lessen,joerg.nitzsche}
@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract Choreographies offer means to describe the long-running collaboration of business partners. Such descriptions can be used to create new participant processes which comply to the overall choreography or to check whether participating processes conform to the protocol. In addition, choreography descriptions allow for asserting whether a completed cross-organizational conversation has been compliant to the planned choreography. However, choreography descriptions have so far not been used during execution but only during design time. Therefore, it is not yet possible to immediately detect protocol violations and to instantly handle such violations. In this paper we motivate the need of a Choreography-aware Service Bus which is capable of tracking the soundness of cross-organizational conversations while they are running. This fosters a novel notion of exception handling in the context of choreographies.

Key words: Compliance, Choreography, Service Bus

1 Introduction

Choreographies offer means to describe the collaboration of business partners. They can be modeled by (i) interconnection models or (ii) interaction models. An *interconnection model* captures the observable behavior of each participant in a choreography. It defines a "network of bilateral interactions", i.e. it defines for each participant in which order it has to send and receive messages to and from its partner services. BPMN [1] and BPEL4Chor [2] are languages to express choreographies by interconnection models using sending and receiving activities. An *interaction model* defines an ordering of process interactions from

* The work published in this article is funded by the SUPER project under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850, http://www.ip-super.org/) and the Tools4BPEL project, which in turn is funded by the German Federal Ministry of Education and Research (project no. 01ISE08).

a global point of view, i.e. it defines a process "in the middle" that captures the interactions of all participants. Current languages providing interaction models are Let's Dance [3] and WS-CDL [4].

In case an interaction model was used to model the choreography, the behavioral description of the participants can be generated. When using an interconnection model, they are already available. These descriptions can be used to either find appropriate participants via conformance checking, or to implement new ones. When a choreography is executed, the choreography itself does not get executed but the participant behaviors described in the choreography are executed. That means, the choreography as an artifact is not used during execution. Since there are constraints that cannot be transformed into local behavior, not all constraints of a choreography can be enforced [5]. An approach is to check the audit log on whether the participants complied with the choreography specification [6]. However, if choreography execution lasts for years and an early message exchange was wrong, it is not desirable to check this choreography violation after the last message. It has to be possible to immediately react on faulty messages to prevent aftereffects.

A Service Bus [7] is middleware to connect services with each other. Services are implementations of participants. A Service Bus provides virtualization of services and thus contributes to a loosely coupled environment. However, besides the lack of a built-in choreography conformance checking, the bus so far does not offer a possibility for a service to register as participant of a choreography and participant virtualization. Such a registration allows for an automatic binding of participants to a choreography. Therefore we propose to make a Service Bus choreography-aware, i.e. to extend it with choreography specific features. For instance, choreography based discovery and conformance checking, which enables the bus to react to violations of the choreography specification in "realtime".

The remainder of the paper is structured as follows: Section 2 shows the state of the art in choreography implementation and describes the limitations of traditional Service Bus when employed for execution. The idea of a Choreography-aware Service Bus is presented in section 3. Section 4 presents related work and section 5 concludes the paper.

2 Limitations of a Traditional Service Bus

Figure 1 presents a sample choreography for investments. A client talks to a financial adviser. The adviser recommends an investment and hands out the requested information material. The government requires to give the customer at least 24 hours to decide on the investment. After 24 hours have passed, the customer may sign the contract. It is not allowed to receive a signature from the customer beforehand.

One approach to implementing a choreography is to transform the choreography to orchestrations [8]. In contrast to a choreography which describes the interaction of all participants from a global view, an orchestration implements the behaviour of a single participant. The Business Process Execution Language

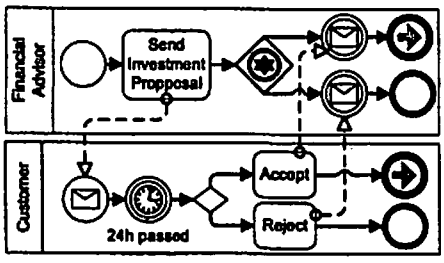


Figure 1. Process of investment offers

(BPEL, [9]) is the de facto standard to describe Web Service orchestrations. Figure 2 presents the steps required to derive executable BPEL processes from a choreography descriptions. These steps are independent from the choreography language chosen. First, the behavior of each participant in the choreography is mapped to an abstract BPEL process. For each participant, the BPEL process is enriched with technical details, such as message types, variables and internal behavior to get an executable BPEL process. These processes then need to be deployed to an infrastructure.

Figure 3 illustrates the scenario for the case in which a traditional Service Bus is used. The Service Bus routes the messages to the corresponding process instance. Each workflow engine generates audit logs. These logs can then be used to check whether the sent messages conform to the choreography. However, a wrong message cannot be held back. If the customer accepts after 12 hours, the bank may continue with the investment. The bank could automatically debit the current account of the customer, transfer the invested money to a third party, etc. If an error is detected by the monitoring application, it could send a message to the bank and exception handling begins. For instance, money transfer has to be compensated.

A crucial point in choreography design is the "local enforceability". A *locally enforceable* choreography is defined as follows: "The global model can be mapped into local ones in such a way that the resulting local models satisfy the following two conditions: (i) they contain only interactions described in the global model;

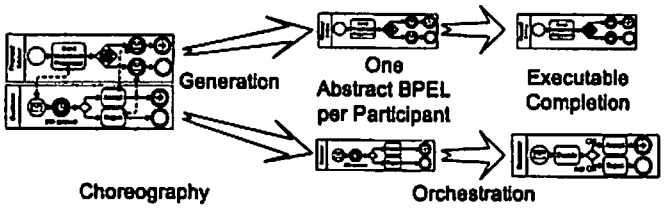


Figure 2. From Choreography to Orchestration

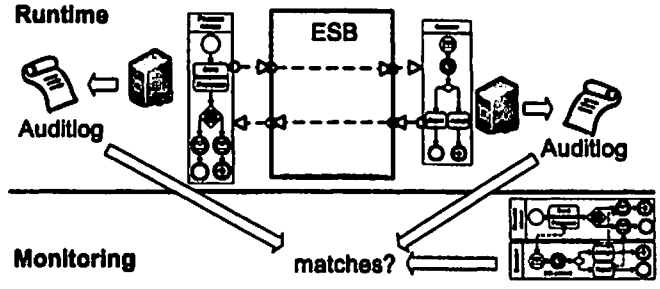


Figure 3. Infrastructure realized with a traditional Service Bus

and (ii) they are able to collectively enforce all the constraints expressed in the global model." [5]. A locally unenforceable choreography is presented in Figure 4: Sender A sends a message to B. Afterwards C sends a message to D. The interaction from C to D must happen after the interaction from A to B. It is not possible to globally enforce a choreography using a traditional Service Bus. The choreography has to be modified to be locally enforceable.

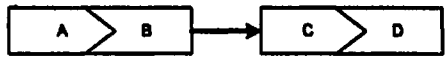


Figure 4. Locally unenforceable choreography [5]

3 Choreography-aware Service Bus

In contrast to the traditional approach, a Choreography-aware Service Bus checks each message to route whether it conforms to the choreography. Figure 5 presents the infrastructure when using a Choreography-aware Service Bus. The choreography-conformance checking is built in: If a message conforms to the choreography, it is routed as in a traditional Service Bus. If a message does not conform to the choreography, the Service Bus does not route the message to the recipient. That means, the wrong message does not trigger any further processing. The bank does not automatically debit the giro account, does not transfer the invested money to a third party, etc.

In general, there are several options to treat a wrong message:

1. Move the message to a dead letter queue
2. Notify the sender that it violated the choreography
3. Trigger default exception handling
4. Trigger a pre-defined exception handling
5. Enforce the choreography

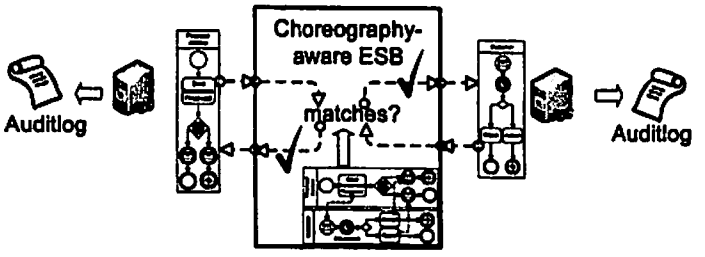


Figure 5. Infrastructure realized with a Choreography-aware Service Bus

Exception handling can include stopping the whole choreography. Another exception handling approach is to only disable messages, which are related to the wrong message. The last option (5) is not applicable in all cases: If the customer agrees too early, the Service Bus can technically hold the message back and resend the message later. However, this is not the intended behavior. In the case of the unenforceable choreography presented in Figure 4, the message sent from C to D can be hold back until the message from A to B has been sent.

4 Related Work

Usually the term "choreography" is used for models decribing the global behaviour of multiple participants. However, sometimes the term "choreography" is used to describe the behavioural interface of a single service, like for instance used in the Web Service Modelling Ontology (WSMO, [10, 11]). The Web Service Modelling eXecution environment (WSMX, [12]) is the reference implementation for WSMO and contains a so called "choreography engine". This choreography engine, however, only supports bilateral message exchanges, i.e. choreographies involving multiple participants are not taken into account.

An overview of languages that enable modelling choreographies involving multiple participants is given in [8]. The service interaction patterns can be used to determine the kinds of service interactions a choreography language supports. While BPEL4Chor and Let's Dance have full support for a priori unknown sets of participants, the possible participants have to be known in advance in WS-CDL [13]. For example, a request-for-bid scenario with a priori unknown bidders cannot be modeled using WS-CDL.

Verification techniques are available for BPEL4Chor [14], BPMN [15,16], Let's Dance [17] and WS-CDL [18]. The verifications ensure that the choreographies are deadlock free or satisfy given properties. The given properties may originate from compliance requirements. Thus, the approaches ensure that the choreography is compliant but do not deal with the realization of a choreography. If a choreography is compliant, a Choreography-aware Service Bus ensures that the participants in that choreography behave according to the choreography and thus are compliant, too.

There are various methods to check conformance between a choreography and orchestrations at design time [19–23]. The drawback of these methods is that at least the behavior description, e.g. expressed in abstract BPEL, of the used services has to be available. If that is not possible, e.g. for services of other companies, the approaches cannot prove whether the opaque service implementation adheres to the choreography specification.

[5, 24] present methods for mapping choreographies to participants such that the participants conform to the choreography. While [24] proposes to ignore unenforceable constraints, [5] requests to reject a choreography which is not enforceable. [25] presents an approach to check whether a choreography is locally enforceable.

Since it is not possible to ensure at runtime that a service implementation conforms to the choreography, runtime monitoring is desirable. [6] presents an approach, where the monitoring data is used to determine whether an obligation has been carried out successfully. The determination is carried out after the execution of the choreography. We presented an approach to ensure conformance at runtime and to prevent false message exchanges.

5 Conclusion and Outlook

We presented how choreography conformance can be realized using a traditional Service Bus. We showed that messages not following the choreography requirements are routed to their destination. A solution is to check the audit logs and to start exception handling if a wrong message was sent.

To overcome the shortcomings of a traditional Service Bus, we presented the concept of a Choreography-aware Service Bus. Using a Choreography-aware Service Bus, messages not compliant to the choreography are not routed to the recipient, but handled in other ways.

Our ongoing work is to evaluate whether existing choreography languages are suitable to be used in a Choreography-aware Service Bus and if there is a need to extend them to specify exceptional behavior.

References

1. OMG: Business Process Modeling Notation, V1.1. OMG Available Specification, Object Management Group (2008)
2. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: ICWS 2007
3. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: CooPIS 2006. LNCS
4. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation, W3C (2005)
5. Zaha, J.M., Dumas, M., ter Hofstede, A.H., Barros, A., Decker, G.: Service Interaction Modelling: Bridging Global and Local Views. EDOC 2006
6. Sailer, M., Morciniec, M.: Monitoring and Execution for Contract Compliance. Technical Report HPL-2001-261, Hewlett Packard Laboratories (2001)

7. Leymann, F.: The (Service) Bus: Services Penetrate Everyday Life. In: ICSOC 2005. LNCS
8. Decker, G., Kopp, O., Barros, A.: An introduction to service choreographies. Information Technology 50(2/2008) (2008)
9. OASIS WS-BPEL TC: Web services business process execution language version 2.0. Technical report, OASIS (2007)
10. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO). W3C Member Submission 3 (2005)
11. Roman, D., Scicluna, J., Nitzsche, J.: D14 V 1.0: Ontology-based Choreography (2007) http://www.wsmo.org/TR/d14/v1.0/.
12. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A semantic service-oriented architecture. ICWS 2005
13. Decker, G., Overdick, H., Zaha, J.M.: On the Suitability of WS-CDL for Choreography Modeling. In: EMISA 2006. LNI
14. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: WS-FM 2007. LNCS
15. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: BPM 2006. LNCS
16. Ghose, A., Koliadis, G.: Auditing business process compliance. In: ICSOC 2007
17. Decker, G., Zaha, J.M., Dumas, M.: Execution Semantics for Service Choreographies. In: WS-FM 2006. LNCS
18. Flavio Corredini, Francesco De Angelis, A.P.: Verification of WS-CDL choreographies. In: YR-SOC 2007
19. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration Conformance for System Design. In: COORDINATION 2006. LNCS
20. Li, J., Zhu, H., Pu, G.: Conformance Validation between Choreography and Orchestration. In: TASE 2007
21. Hongli, Y., Xiangpeng, Z., Chao, C., Zongyan, Q.: Exploring the Connection of Choreography and Orchestration with Exception Handling and Finalization/Compensation. In: FORTE 2007. LNCS
22. M.Bravetti, Zavattaro, G.: Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In: 6th International Symposium on Software Composition (SC'07). LNCS
23. Aalst, W., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, H.M.W.: Choreography conformance checking: An approach based on BPEL and petri nets (extended version). BPM Center Report BPM-05-25, BPMcenter.org (2005)
24. Mendling, J., Hafner, M.: From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In: MIOS 2005
25. Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: BPM 2007. LNCS

All links were last followed on May 8, 2008.

Web Oriented Integration Architecture for Semantic Integration of Information Systems

Daniel Szepielak, Przemyslaw Tumidajewicz

Deutsches Elektronen-Synchrotron DESY, Notkestrasse 85, 22607 Hamburg, Germany
daniel.szepielak@desy.de, przemyslaw.tumidajewicz@desy.de

Abstract. With the growing popularity of the Web 2.0 concept a subtype of Service Oriented Architecture called the Web Oriented Architecture (WOA) is gaining increased attention. WOA simplifies the model of interaction for networked applications and introduces a uniform functional interface for accessing information system resources. The following paper describes the Web Oriented Integration Architecture (WOIA) which extends the WOA with semantic integration capabilities. The WOIA allows information consumers to access information system resources regardless of their representations and interfaces of particular information systems, as well as without knowing where the required resources are located. The necessary functionality is delivered by the WOIA middleware which provides web service registration, discovery, composition and execution capabilities. Development of necessary web services is facilitated by a proposed REST web service framework.

Keywords: Web Oriented Architecture, REST, Web 2.0, semantic integration

1 Introduction

The issue of information systems integration in enterprise environments has posed a big challenge in the domain of software development and maintenance since more than two decades. Modern enterprises use large sets of information systems which constantly evolve as new information systems and services are introduced and legacy systems are replaced. Maintaining such ever-changing environments requires constant monitoring of the integration infrastructure and considerable amounts of development work whenever a change occurs [1].

This paper proposes a conceptual model for semantic integration of information systems. It aims at an integrated environment which is capable of discovering and accommodating a growing number of systems and services within a common semantic context. Semantic integration [2][3] refers to a procedure that allows information systems to integrate without human intervention as long as they share a common understanding of their business concepts.

2 Background

Recent years have witnessed an unprecedented shift in distributed computing towards Service-Oriented Computing (SOC) [4] which is gaining prominence as an efficient approach for integrating applications in heterogeneous distributed environments [5][6]. The most popular branch of SOC research is dedicated to advances in Service Oriented Architecture (SOA) [7] and SOAP [8] web services, but the growing popularity of the Web 2.0 [9][10] concept has brought increased attention to the Web Oriented Architecture and REST web services [11] as an alternative way of building service oriented environments.

Web Oriented Architecture (WOA) is a term coined by Nicholas Gall of Gartner Research as an architectural sub-style of SOA based on the architecture of the WWW. WOA is essentially a lightweight version of SOA that in many ways follows the principles of Web 2.0 and strongly promotes simple, open and web-scale approaches. The major goal of WOA is to enable the development of globally linked, decentralized systems that have the flexibility of the Web and offer uniform intermediary processing of application state via self-describing messages.

SOA and WOA are consistent at the conceptual level. At implementation level, SOA is based on SOAP web services and follows the distributed object approach with object-specific methods and remote procedure calls. WOA follows the philosophy of the Web and employs REST web services to achieve a single uniform functional interface for all kinds of objects using open standards only.

REST is the acronym for Representational State Transfer defined by Roy T. Fielding [12], a founder of the Apache Software Foundation and one of the authors of the HTTP protocol specification. REST describes an architectural style of networked systems designed around the concepts of a resource (typically represented as an XML document) and a uniform resource identifier (URI). REST is commonly used together with the CRUD pattern [13] which defines the basic set of atomic operations: Create, Retrieve, Update and Delete. These operations are mapped to HTTP protocol methods (as shown on Fig. 1) and can be performed on any resource, allowing the consumers to manipulate all resources in the same consistent way. Each CRUD operation in this approach can be seen as an individual web service.

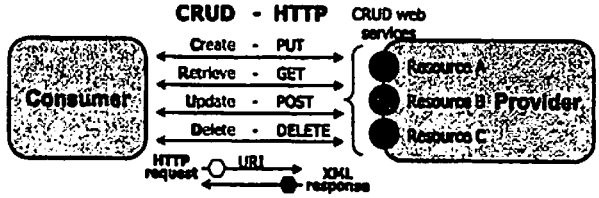


Fig. 1. Interaction of WOA components using REST web services

The uniformity of the REST interface implies that no additional description of a web service signature and behavior is necessary. The use of the CRUD pattern implies that providers and consumers need to share a common understanding of the syntax and semantics of a message. This can be achieved with a wide range of technolo-

gies such as Document Type Definition (DTD) [14], Resource Description Framework (RDF) [15] or Web Ontology Language (OWL) [16].

Another important characteristic of WOA and REST is that resource representations can include hyperlinks to further related resources. This allows consumers to navigate sets of resources without prior knowledge of possible resource interconnections. For example, the representation of a *Component* resource may contain links that point to its *Requirement* resources. The requirements may be captured in a technical specification *Document*, which again describes the *Component* resource. A further link could point to the *Location* at which the *Component* is installed (Fig. 2).

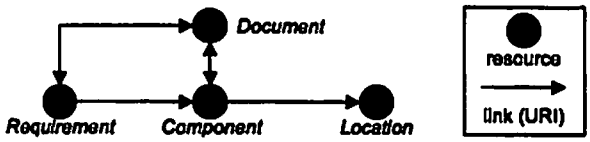


Fig. 2. WOA resource set example

3 Addressing the Challenges of WOA Based Integration

Many existing WOA applications are limited to directly connecting resource providers and consumers. This situation results from the simplicity of WOA, which is considered to be its greatest advantage, but at the same time restrains its usage in more complex integration scenarios. Basic principles of WOA work well as long as all the resources are controlled by a single provider. In case of enterprise information systems integration, the conceptual enterprise business model is decomposed into fragments realized by individual information systems and resources distributed across multiple providers. This difficulty can be addressed by providing a common view of the business model in form of an ontology which contains the necessary resource definitions and their relationships. Careful consideration of semantic integration requirements and their confrontation with characteristics of the WOA yields the following challenges that need to be addressed:

Consumers should be able to discover resource URIs. Mechanisms should be introduced which allow consumers to discover the URIs of resources from a central point of where providers publish information about the resources they are offering.

Consumers should be able to access business meaningful resources using a single URI. Each information system in an integrated environment may offer web services that allow consumers to access its ***actual resources*** **(AR)**, i.e. resources as they are maintained in the system. These ARs often hold only partial information about higher level conceptual resources, because higher level resources may have been split into several ARs in different information systems. Actual resources need to be composed according to the ontology into ***business meaningful resources*** (BMR) which reflect objects from the business model.

Consumers should be isolated from changes of resource URIs. URIs pointing to actual resources are subject to occasional changes (URI instability is one of the major risks of REST). Each resource URI change forces the consumer to adapt, so from the

consumer's point of view, a resource URI should ideally never change. This can be achieved by mapping providers' *actual URIs* pointing to ARs to stable *virtual URIs* which address BMRs. In case when a BMR is composed of more than one AR a *virtual URI* will be mapped to many *actual URIs*.

Providers should not be responsible for providing links between business meaningful resources. In most cases, providers will only have knowledge of a subset of the company business model that corresponds to their own platform specific model. They may therefore be incapable of identifying relations at the BMR level. This makes it necessary to shift the responsibility for providing links among BMRs to an external component which has access to the complete business model.

The simplicity of using REST web services has to be preserved. One of the main reasons for choosing WOA and REST is their simplicity. Extending the base capabilities of WOA should not be done at the cost of simplicity as this would invalidate one of the key requirements.

4 Web Oriented Integration Architecture

The Web Oriented Integration Architecture (WOIA) [17] addresses the challenges by providing a uniform and lightweight access interface to BMRs defined in an ontology which represents the unified business model. This separates ever-changing platform specific concerns from high level conceptual decisions and allows the information systems to maintain platform specific models while making resources accessible in a platform independent manner. Fig. 3 gives an overview of the integration architecture aligned to the traditional three-tier information system architecture.

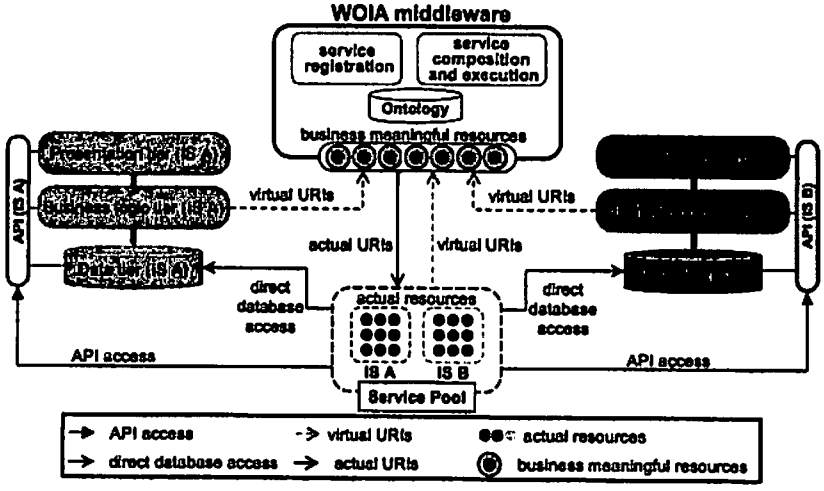


Fig. 3. Web Oriented Integration Architecture (WOIA) overview

The middleware component introduced by the proposed architecture allows the information systems within an integrated environment to access each other's resources via the REST interface. This makes it easy and straightforward for the developers to use the resources without the need of learning system specific APIs. The concept of the middleware is based on combining data about web services and their providers with an ontology that represents the BMRs. ARs provided by the information systems are registered with the middleware component along with their mappings to concepts defined in the ontology. As a BMR may consist of many ARs delivered by various providers, the middleware component can also compose, execute and coordinate multiple web services for a single consumer request, if necessary.

As the middleware component exposes a REST based interface for the registered BMRs, accessing them becomes as simple as accessing a resource URI, like in the original WOA. The entire task of translating a request for a BMR to a set of requests for ARs, executing those requests and building an aggregated response for a consumer is handled by the middleware component. This way, the architecture fulfills the goal of presenting consumers with a lightweight resource access model.

As the middleware is the only component which has access to the complete business model, it is also responsible for providing links to related resources. Available links are determined basing on relations between BMRs as defined in the ontology. This way, the architecture allows for effective management and coordination of the web services delivered by the different providers.

The proposed architecture can also be used to handle process based resources. This requires translating functional aspects of a process into a data structure that can be represented in an ontology. Such translation can be achieved by defining input and output parameters of the process as attributes of an ontology class and using the standard CRUD operations to start a process (*Create*), to check the process status (*Retrieve*), to alter or interact with a process (*Update*) and to abort or terminate the process (*Delete*).

5 Framework for REST Web Service Based Integration

Building an integration solution based on the WOIA architecture requires developing families of REST web services for all the resource classes defined in the ontology. The development process can be considerably sped up by providing a framework for building the web services. The goal of the framework is to increase code reusability, and to ease maintenance by collecting code that is shared among the web services. The proposed framework utilizes the following identified sources of reusability.

General code shared among all web services, captured in the *Request-Response Processor*. Regardless of the specific functionality of a particular web service, general tasks such as parsing requests and configurations, handling exceptions, formatting responses etc. have to be performed by all web services.

Code shared by web services which are performing the same operation, captured in the *Operation Controller*. Every resource is providing a CRUD interface, implying that e.g. *Update* services have to be implemented for a variety of resources. Regardless of the resource type, the same operation always follows the same reusable

sequence of atomic actions. For instance, every *Update* operation requires the service to: open a transaction; find a resource using a given set of filters; if the resource exists, update it with provided attribute values; commit the transaction on success or rollback otherwise.

Code shared by web services which are connecting with the same information system, captured in *System Resource Drivers*. Implementation of atomic system interactions like connecting and disconnecting from a system, beginning or finalizing transactions, creating, locating, retrieving, modifying and deleting resources do not change between services operating on the same system.

Fig. 4 presents the framework architecture. An example resource driver hierarchy illustrates the case of two information systems, an Engineering Data Management System (EDMS) and an Inventory Management System (IMS), which are offering access to *Document*, *Component* and *Location* resources.

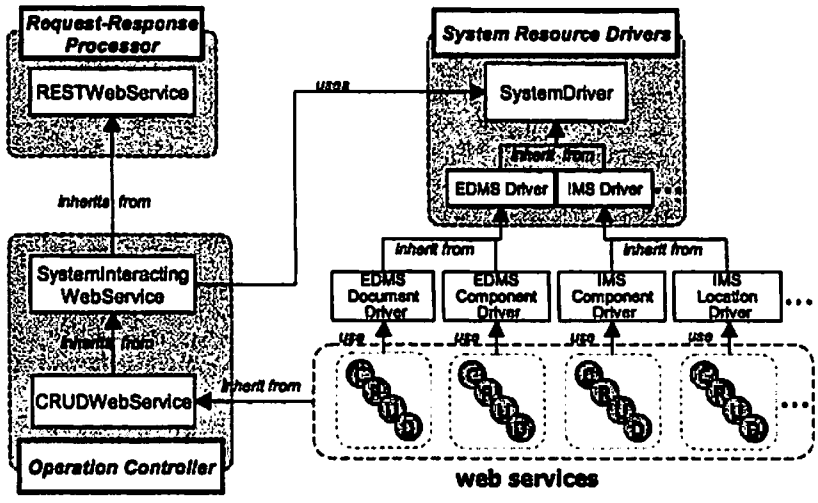


Fig. 4. Framework structure

The left side of Fig. 4 shows components which implement the operational skeleton of a web service, while the right side represents information system specific code. The *RESTWebService* class realizes the *Request-Response Processor* functionality and is responsible for handling incoming requests and passing their parameters and stored configuration to the *SystemInteractingWebService* class for further processing. Upon successful completion *RESTWebService* formats and returns the results. In case of failure, it handles the exceptions and returns error messages.

The *SystemInteractingWebService* and *CRUDWebService* classes realize the *Operation Controller*. The *SystemInteractingWebService* class encapsulates system interactions common for all web services. It creates an instance of a *SystemDriver* and uses it to open a connection to the information system and handle transactions. The connection is passed to the *CRUDWebService* class for operation specific processing.

The *CRUDWebService* class dispatches CRUD operations depending on the HTTP method used for the request and executes each of the four basic operations as a sequence of atomic system interactions.

The *SystemDriver* class provides an interface for the enclosed set of atomic system interactions and acts as the parent class for all specific information system drivers. As each information system can provide various resources for which the behavior of the driver may be partially different, drivers for specific information systems can have sub-hierarchies of resource specific drivers.

The web service classes for each resource type extend the *CRUDWebService* class, thereby inheriting the full operational skeleton, and use *SystemDriver* subclasses to deliver the atomic operations for the particular system-resource combination.

6 Results, Conclusions and Future Work

The presented WOIA integration approach has been developed and applied in the engineering data management domain at the Deutsches Elektronen-Synchrotron (DESY) in Hamburg, Germany. DESY is one of the world's leading centers for research with particle accelerators. DESY develops and operates large scale accelerator facilities and conducts basic research in a great variety of scientific fields, ranging from particle physics to materials science and molecular biology.

Web services developed for the DESY integrated environment allow for accessing resources like e.g. different types of documents, accelerator components, person information, locations and more. The number of implemented web services can be kept moderate because a single web service is often used for accessing multiple AR classes. The resources are distributed across an Engineering Data Management System, an Inventory Management System, a Facility Management System and a Personnel Information Database.

All WOIA components have been implemented using Java. The solution has proven to be a flexible and powerful integration platform. WOIA reduces the necessary human involvement to the essential integration tasks of creating a business model and providing web services for basic information system access. Using uniform REST web services to access to information systems' resources has proven to be genuinely useful, especially for applications which are cooperating with multiple information systems. The observed average level of code reuse per web service is 93%. This figure is based only on the internal reuse of the proposed framework code and does not count external libraries or parts of the framework that are not used by particular web services.

Although the presented integration approach has been developed to fulfill specific DESY requirements, the achieved solution is of general purpose and can be deployed in other typical information system environments with little or no adjustments. Applying the WOIA approach requires developing an ontology that will represent the business model of the target organization. Subsequently, a set of web services for accessing the ARs of the organization's information systems has to be provided – or, if suitable web services are already available in the environment, their interfaces need to be mapped to the basic operations on resources defined in the ontology.

As the major goals of the project have been achieved, follow-up issues for further research and development are emerging. Introducing distributed transaction handing and addressing the issue of Single Sign-On would offer useful extensions to the resource access mechanisms built into the WOIA middleware. Furthermore, XML encryption could be explored as a way of increasing security measures in critical applications of the proposed solution.

References

1. Hohpe G., Woolf B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (2003)
2. Gruninger M., Kopena J.: Semantic integration through invariants. In Proceedings of Workshop on Semantic Integration at ISWC-2003, Sanibel Island, FL, (2003)
3. Noy N.F.: Semantic integration: a survey of ontology-based approaches. ACM SIGMOD Record Volume 33. pp. 65-70 ACM Press, New York, NY, USA (2004)
4. Chang M., He J., Castro-Leon E.: Service-Orientation in the Computing Infrastructure. In Proceedings of second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)
5. Mayerl C., Vogel T., Abeck S.: SOA-Based Integration of IT Service Management Applications. In Proceedings of IEEE International Conference on Web Services (ICWS'05)
6. Erradi A., Anand S., Kulkarni N.: Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. In Proceedings of IEEE International Conference on Services Computing (SCC'06)
7. Krafzig D., Banke K., Slama D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall (2004)
8. Newcomer E.: Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley Professional, 1st edition (2002)
9. Millard D.E., Ross M.: Web 2.0: hypertext by any other name? In Proceedings of the seventeenth conference on Hypertext and hypermedia. HT'06 Odense, Denmark, pp. 27-30
10. Musser J. and O'Reilly Radar Team: Web 2.0 Principles and Best Practices. ISBN: 0-596-52769-1 O'Reilly Radar (2006).
11. Vinoski S.: REST Eye for the SOA Guy. IEEE Internet Computing, vol.11, no.1, pp. 82-84, (2007)
12. Fielding R.: Architectural Styles and Design of Network-based Software Architectures. PhD Thesis, UC Irvine (2000)
13. Caserta J., Kimball R.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons, New Yersey 2004
14. Lv T., Yan P.: Mapping Relational Schemas to XML DTDs with Constraints. In Proceedings of First International Multi-Symposiums on Computer and Computational Sciences - Volume 2 (IMSCCS'06) pp. 528-533
15. Brickley D., Guha R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, (2004)
16. Bechhofer S., Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A.: OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/
17. Szepielak D.: Web Oriented Integration Architecture for semantic integration of information systems. PhD Thesis, Silesian University of Technology, Gliwice / DESY, Hamburg (2007)

ReforMing Mashups

Ziyan Maraikar[1] and Alexander Lazovik[2]*

[1] Centrum voor Wiskunde en Informatica, The Netherlands
Z.Maraikar@cwi.nl
[2] INRIA Saclay, Parc Club Orsay Université, France
lazovik@lri.fr

Abstract. The explosive popularity of mashups has given rise to a plethora of "mashup platforms". Using these web-based tools, mashups can be rapidly constructed with minimal programming effort. Reo for Mashups (ReforM) is a service composition platform that addresses service heterogeneity as a first-class concern, by adopting a mashup's data-centric approach. Built atop the Reo coordination language, ReforM provides tools to combine, filter and transform web services and data sources like RSS and ATOM feeds. Whereas other mashup platforms intermingle data transformation logic and I/O concerns, we aim to clearly separate them by formalising the coordination within a mashup. Reo's well-defined compositional semantics opens up the possibility of constructing a mashup's core from a library of prebuilt connectors. We believe these are compelling features as mashups graduate from curiosities on the Web to situational applications for the enterprise.

1 Introduction

A recent trend in web applications has been the emergence of so-called *mashups*. Mashups are web applications that literally mash-up or combine disparate web services, RSS and ATOM feeds and other data sources in new and interesting ways. They compose these services in ways usually unanticipated by their original authors. Mashups are thus ad-hoc by very nature.

Many approaches to service-oriented computing (SOC), in contrast, assume the availability of uniform service interfaces and service metadata available in centralised registries. In reality, the SOAP and WSDL web services standards promulgated by the W3C have seen limited adoption outside of closed enterprise environments. Neither has the promise of universal service registries, in the guise of UDDI or otherwise, been realised.

Prominent web service providers like Google, Yahoo, Amazon and eBay have opted to use lightweight protocols like RSS and ATOM to push data to consumers, while exposing their service offerings as simple REST-style[5] APIs. In turn each of these service provider APIs have their own syntax and semantics.

* This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme

A realistic approach to SOC on the Web, must therefore address service heterogeneity as a first-class concern, instead of relegating it to a mere implementation detail.

Reo for Mashups (ReforM) is an attempt to bridge the gap between the restrictive homogeneity assumptions of the more formal approaches to SOC and the completely ad-hoc nature of mashups. Where mashups succeed is in adopting a data-centric approach to service composition. In a sense, they follow the UNIX tradition of gluing arbitrary programmes together using pipes and text processing tools like *awk* and *sed*, to meet any need. In similar spirit ReforM provides tools to graphically define a mashups' data-driven logic, and plug services together with precise formal semantics. We believe this is a useful approach that complements the traditional control-driven coordination and orchestration exemplified by BPEL.

1.1 Mashup Platforms

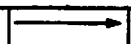
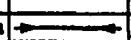
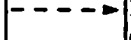
Mashups can, and are, built using traditional web scripting languages like Perl, PHP and JavaScript. Their explosive popularity has however, given rise to a number of "mashup platforms" for building mashups, with minimal programming. This trend parallels the rise of rapid application development (RAD) tools to ease the development of graphical user interfaces in the 90's.

The use of mashups as *situational applications* in enterprise environments is addressed in [6]. They use the phrase "Enterprise information mashup fabric" to describe mashup platforms for the enterprise. Damina[1] describes a concrete realisation of these ideas for building mashups on corporate intranets. Both ReforM's tools and runtime environment have similarities to Damia. However, we are more concerned with enabling compositional construction of mashups with well defined semantics. In this respect the ReforM tools share a common purpose with the SENSORIA Development Environment[9] (SDE) for semantic-based development of service-oriented systems. Whereas SDE is general purpose SOC tool suite, ReforM focuses on exclusively on tools for data-driven service coordination.

ReforM also borrows ideas from commercial mashup platforms. Google mashup Editor[3](GME) is a textual mashup tool that uses XML based mark-up, combined with HTML and optionally, JavaScript. Each user-interface element in the mashup is defined using a `module` tag that groups an input data source and a template that formats resulting output. Yahoo Pipes[4], another commercial offering, takes a graphical approach to mashup creation. A variety data sources can be plumbed through so-called *pipes* that filter and transform data. Predefined pipes are available to connect to data sources like RSS feeds, REST and SOAP web services, perform string, number and date manipulation, and filter and sort data. Complex pipes may be composed of primitives and invoke external services. Pipes superficially resemble channels in the Reo coordination language

[3] `http://googlemashups.com`
[4] `http://pipes.yahoo.com`

Table 1. Behaviour of common Reo channels

Sync		Simultaneously accepts data on one end and passes it out the other end
SyncDrain		Simultaneously accepts data on both ends
LossySync		Behaves as a *Sync* if a *take* operation is pending on the output end, otherwise the data is lost
Filter		Passes data matching a filter pattern and loses the rest
FIFO		Buffers a single data item.

(see §1.2). Coincidentally, ReforM's editor closely resembles Yahoo Pipes, but the Reo tool suite described in §3 actually predates it.

Most mashup platforms lack a clear separation between the user interface and program logic. Almost none of them consider the semantics of the services being used. Consequently none support true compositional construction of a mashup's logic. Although Yahoo Pipes supports composing pipes, its notion of composition is ill-defined. One might counter that these issues are largely irrelevant as today's mashups are mostly toy applications. As mashups migrate into enterprise environments, however, a more disciplined approach becomes highly desirable. ReforM aims to provide the features necessary to support this use case, without unduly burdening the mashup developer.

1.2 Reo

We use Reo[2] as the basis for expressing coordination in a data-centric fashion. Extending our UNIX tools analogy, Reo can be viewed as a generalisation of UNIX pipes. Reo is *exogenous* in that it imposes a coordination pattern on components, without any knowledge of the specifics of the components themselves. This makes Reo ideal for coordinating services from a data-centric perspective. We give only an informal overview of Reo below. The canonical reference for Reo is [2], and its semantics expressible as constraint automata or connector colouring is detailed in [3, 4] respectively.

Coordination in Reo is specified by a *connector* — a graph consisting of primitive *channels* and *nodes*. A *channel* has two ends, which may be sources or sinks, on which input/output is performed. Its behaviour is specified by synchronisation and data constraints. Table 1 describes the behaviour of a subset of common Reo channels. Channel ends are linked together by *nodes* that do not buffer data. Nodes can replicate the source channel-end across several input-ends or merge the sink ends of several channels onto a single source end. Components communicate with Reo via a connector's *ports* which correspond to its source and sink nodes — those having no incoming or outgoing channel ends, respectively. An example of a Reo connector is described in Section 2.1.

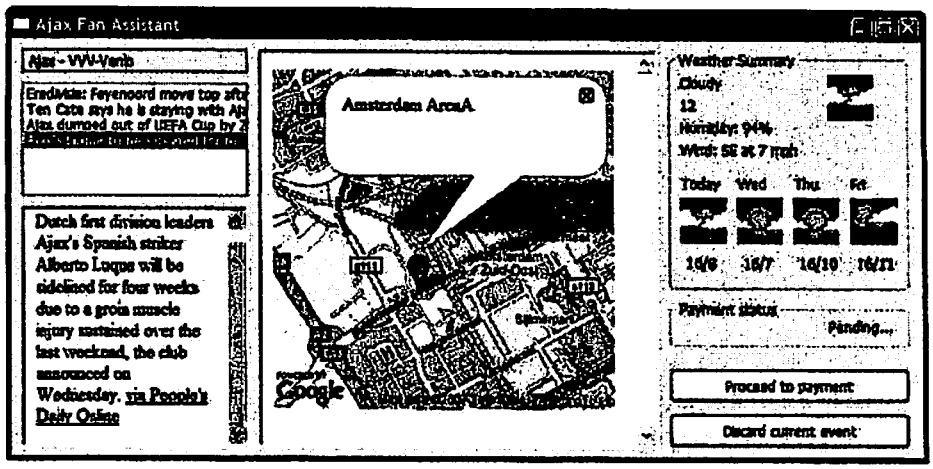


Fig. 1. Example sports-fan mashup user interface

2 ReforMing Mashups

ReforM attempts to bridge the gap between classical SOC and mashups by imposing a structured approach to building mashups that does not sacrifice their flexibility. We begin by framing the steps involved in building a mashup in SOC terminology, and identifying possible improvements to the process.

Service binding There is an inherent trade-off between flexibility in service binding vs. dealing with service heterogeneity. The late-binding approach advocated in SOC assumes uniform service interfaces, and universal registries. Conversely, mashups handle heterogeneity at the cost of being tightly coupled to the services they use. Ideally we would like a limited form of late-binding at least for standard input source like RSS feeds.

Composition and coordination Control-driven coordination as exemplified by BPEL is the norm in SOC. On the other hand, coordination in the context of a mashup usually boils down to ad-hoc "data mangling"; that is, filter combine and transform various inputs to generate desired outputs. Furthermore, a mashup's logic should be composable from reusable blocks to facilitate quick, modular construction. Yahoo Pipes for instance does permit composing pipes, but the notion of composition should be formalised to ensure correctness.

User interaction SOC regards user interaction as an external concern. In a mashup there is a direct link between the data mangling logic and user interface elements that render the data as text or graphics, e.g. a map with pushpins. It is highly desirable to have a clear separation between coordination logic and user interaction aspects of development.

We use a "Sports Fan" application to demonstrate mashup development in ReforM's, and highlight how we achieve the potential improvements identified above. The "Sports Fan" mashup shows relevant information based on the fixtures of a sports team. The team publishes a feed of their calendar containing a list of fixtures e.g., A fixture calender for Ajax FC is available on Google Calendar[5]. Once the user chooses a match of interest we display the following information: (i) a map showing the venue; (ii) news articles about the fixture; (iii) weather forecast for the day of the fixture; (iv) option to purchase tickets online, if the weather forecast is "good"[6]. Figure 1 shows a screenshot of the running application.

2.1 Coordination in ReforM

We formalise the notion of *coordination* within a mashup by encoding it in Reo. Augmenting the Reo tool suite with filter and transform channels, gives ReforM functionality common to other mashup platforms. Filter channels take a filter expression e.g. a data type or regular expression to match. If a datum matches the filter expression it passes through the channel, otherwise it is dropped . A transform channel, likewise, accepts a data transformation expressed e.g. as a *sed*-like replacement or XSLT. Each input datum is rewritten according to the transform expression as it passes through. Filters and transforms can also execute user-defined functions. For example, a geo-coding channel that converts place names to latitude and longitude could invoke an external service. Semantically a filter acts as a specialised *LossySync* channel while a transform acts as a *Sync*.

Figure 2 shows the coordination in our sports fan application defined in Reo. A sync channel is used to display the feed on the user interface. We use two channels to extract the venue and names of the teams from the feeds. These are fed to the Map and News components respectively, displaying the currently selected match venue and relevant news items. The extracted venue is also used to retrieve a weather forecast, if available. A filter channel classifies whether the weather is "good", and if so we give the user the option to purchase a ticket online.

Using Reo as the basis for coordination offers a number of advantages. The Reo connector in Figure 2 completely specifies not only the data mangling in the mashup, but also any synchronisation required between individual services. Furthermore, a mashup's coordination logic could be composed out of a library of predefined connectors. Unlike say, Yahoo Pipes, these connectors can be composed with precise semantics, thanks to Reo's compositionality.

3 Implementation

Our mashup design tools consist of enhancements to the Eclipse Coordination Tools[7] (ECT) — a suite of graphical tools for Reo, and a runtime environment

[5] `http://www.google.com/calendar/embed?src=jdtvbcrk1vtpn5ec4ptnnfnd61c0udfp%40import.calendar.google.com`

[6] Admittedly this was contrived to show the utility of channel composition in Reo

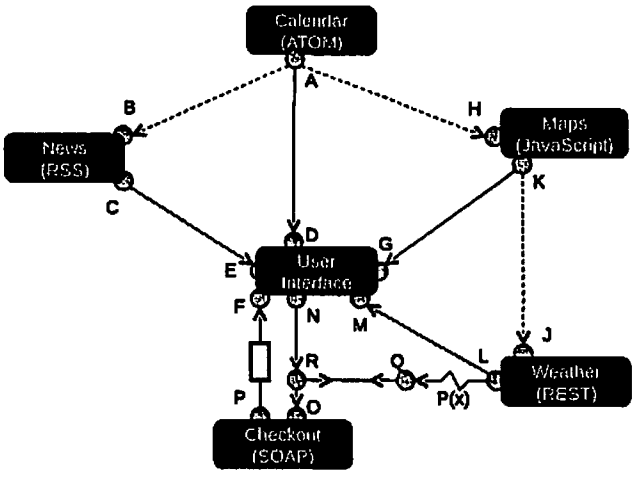


Fig. 2. Reo connector implementing Sports fan mashup's coordination

for mashup execution. Built on the Eclipse Graphical Modelling Framework, ECT consists of graphical editors for Reo connectors, an animator for visualising a connector's semantics, and transformation from Reo to constraint automata. With the addition of Filter and Transform channels ECT's Reo editor achieves feature parity with graphical mashup builders like the Yahoo Pipes editor.

ReforM's execution environment depicted in Figure 3, consists of a Reo engine and a management interface, hosted in a servlet container such as Tomcat. Reo has various executable implementations which may be used to run a ReforM mashup. ReoCP is a constraint programming engine that directly executes a Reo circuit based on colouring semantics for Reo. CASP generates Java code from constraint automaton representation of a Reo connector. Work is also underway on a distributed Reo implementation[8] on Scala Actors. Any one of these engines may be plugged into the ReforM runtime via a common interface.

The management interface lets a user deploy Reo connectors and start and stop connector instances via a web browser. When a connectors is deployed and started the runtime initialises the Reo engine with the given connector and instantiates server-side wrapper and user interface components. Wrappers (pink) an UI components (orange), depicted in Figure 3, communicate with the engine via read and write operations on ports (grey) of the Reo connector being executed.

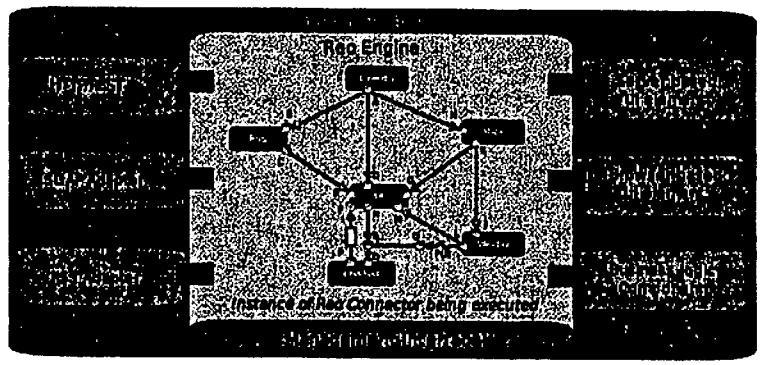


Fig. 3. The ReforM Runtime environment

3.1 Interacting with The World

Input/output considerations are an integral part of mashup design. The ReforM runtime exposes ports of a Reo connector being executed using a synchronous read/write protocol. Ports are also mapped to URLs by the runtime. Remote components may read and write data to ports using the HTTP GET and PUT operations on these URLs.

ReforM uses thin wrappers to bind to services and feeds. Wrappers simply bridge remote services and Reo ports. Similar to Yahoo Pipes, we envisage a library of common wrappers for standard data sources like RSS feeds and SOAP web services. Outputs generated by a mashup are displayed by user interface elements — anything from a simple HTML list to a map overlaid with complex data. A UI element either executes locally on the same server as the mashup, or remotely on the user's browser such as a component using the Google Maps JavaScript API. Such browser-based UI elements may use JavaScript's `XMLHTTPRequest` object to perform I/O via the URL mapped ports, with the Reo connector executing on the server[7].

4 Conclusion and Future Work

We have presented a framework for composing heterogeneous data and services on the Web. Rather than approach service coordination from a control flow perspective, we take a data-centric view inspired by mashups. By using Reo as the basis for formalising coordination logic in a mashup, we improve on the current state of the art in platforms for mashup construction. Our tools and techniques introduce a clear separation of concerns and permit compositional construction of mashups with precise semantics, without sacrificing the traditional mashup's strengths of rapid development and flexibility.

[7] This is same the technique known as *AJAX* in common parlance.

The synchronous semantics of Reo gives us simple transactions that can ensure that a chain of components connected by channels all execute atomically. Component failure is however, not dealt with at present[8]. In our current implementation a filter channel applies a filter expression as data passes through it. For better efficiency on large data sets we hope to implement a technique for to pushing these queries back through the connector to services or their respective wrappers. Reo has also gained the ability to dynamically reconfigure connectors, which opens up the possibility of adapting connectors at runtime, to facilitate late-binding of services for instance.

References

1. M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y. Ng, D. Simmen, and A. Singh. Damia - a data mashup fabric for intranet applications. In *VLDB*, pages 1370–1373, 2007.
2. F. Arbab. Reo: a Channel-based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14:329–366, 2004.
3. C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in Reo by Constraint Automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.
4. D. Clarke, D. Costa, and F. Arbab. Connector colouring I: Synchronisation and context dependency. *Electr. Notes Theor. Comput. Sci.*, 154(1):101–119, 2006.
5. R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. Chair-Richard N. Taylor.
6. A. Jhingran. Enterprise information mashups: Integrating information, simply. In *VLDB*, pages 3–4, 2006.
7. C. Koehler, A. Lazovik, and F. Arbab. Reoservice: Coordination modeling tool. In *Conf. on Service-Oriented Computing (ICSOC-07)*, Lecture Notes in Computer Sciences 4749, pages 625–626. Springer, 2007.
8. José Proença. Towards Distributed Reo. Talk presented at CIC workshop, 2007. `http://homepages.cwi.nl/~proenca/distributedreo`.
9. Martin Wirsing, Allan Clark, Stephen Gilmore, Matthias Hölzl, Alexander Knapp, Nora Koch, and Andreas Schroeder. Semantic-Based Development of Service-Oriented Systems. In E. Najm et al., editor, *Proc. 26th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems(FORTE'06), Paris, France*, LNCS 4229, pages 24–45. Springer-Verlag, 2006.

[8] Work is ongoing to add compensation to Reo

Analysis of a Federated Identity Management Protocol in SOC

Qurat-ul-Ain Nizamani, Hyder Ali Nizamani

Department of Computer Science, University of Leicester, UK
{qn4,han3}@le.ac.uk

Abstract. Service-oriented computing allows inter-organisation collaboration and data exchange in a standard dynamic way. In order to achieve such collaborations, organisations need to establish identity and trust relationships with partner organisations. Federated Identity Management (FIM) yields such notions by using *Identity Service* as a way to exchange security related information between the identity provider and service provider. Various protocols have been defined to develop such services in a service-oriented approach. In this paper we formally verify a FIM protocol with the help of a symbolic model checker.

1 Introduction

Service-oriented computing (SOC) provides platform as well as language neutral, loosely coupled and highly interoperable characteristics for rapid application integration. Moreover, with SOC, inter-organisation collaboration and data exchange can be done in a standard dynamic way. To protect this interaction in a SOC environment, security measures must be applied to ensure authentication, namely that e.g., requesting party is actually the intended one and not a competitor or spy. This can be achieved by providing identity and trust mechanisms.

For organisations to establish identity and trust relationships, they have to sign contractual agreements, privacy laws, and liabilities with their partner organisations. Once they establish such trusted relationships, technical problems to manage identity and trust could be suitably tackled using an identity management infrastructure.

Federated identity management (FIM) provides an infrastructure for enabling authentication through an identity service capable to enforce security across different domains [5]. In a standard based FIM approach, an identity provider (IP) is considered a fundamental component which would be responsible for exchanging security related information (i.e., authentication and access control) between cooperating organisations or different applications operating within the same organisation. So, the IP exposes identity services to exchange security related information in a way that service providers can find and bind such services in a dynamic way. This parallels what SOC provides for dynamic application integration. Interoperability can be indeed obtained by using the SOC paradigm and its related technologies (e.g., HTTP, XML, SOAP, UDDI and WSDL) when developing identity services [5–7].

FIM infrastructure manages user identities across different domains, which enables individuals to interact with a service provider (SP) with trust relationships by signing in just once and without storing identities centrally. This notion supports a technique called *Single Sign-on* (SSO) in which a user authenticated at local organization is allowed to access resources across the organisations without requiring re-authentication. Fig.1 depicts a SSO scenario where the users of a domain (i.e., travel.com) can access services provided by other domains (i.e., hotel.com and carRentals.com) without requiring re-authentication. In the above example, users share their identities between the organisations which follows the specifications of [5] operating within a *Circle of Trust* (CoT). A CoT is defined as a group of SPs that share linked identities at a single identity provider (IP) and have pertinent business agreements in place, regarding how to do business and interact with identities in a secure and apparently seamless environment. A user authenticated by a CoT IP, can access and consequently take part in targeted services from other SPs within the CoT [8]. The system that initially authenticates the user is responsible for passing the authentication information (i.e., user's identity and attributes) while user requests other services.

In traditional FIM, the concept of CoT is static as business and security policies are decided in advance to become a part of CoT. In a SOC scenario it is important to share service identities, so that the services authenticate themselves with other services in a dynamic way and this can be achieved in a similar fashion as FIM does for sharing user identities. For example, an online book shop runs a book purchasing service which looks for various book seller services and dynamically finds them on the internet (i.e., amazon.com and xyz.com). After a service selection on a certain criteria like quick, cheap and reliable shipment, the book purchase service decides to place the order at amazon.com. We assume that the order placement service at amazon.com requires authentication of the calling service along with payment details. In this case, the service authentication requires certified service attributes (i.e., name, address and bank acount details of the online book shop) which should be verified by an online certification authority (CA). It is required that CA must be a trusted one by the service provider so that a dynamic CoT can be formed to let service consumers - in this case the book purchasing service - access the services. The CA is expected to share service attributes with amazon.com by exposing an Identity Service from its IP which needs to be more flexible so that SP can dynamically search and bind service identity interfaces and security policies through registry services.

In this paper, we formally analyse a FIM protocol using symbolic model checking. The protocol has been proposed in [9] and studied in [3]. Security protocols like FIM are run on public network and therefore are vulnerable for various attacks by intruders, making the protocol incapable of holding security properties like integrity, authentication, controlled access to services or resources and so on. It is therefore important to formally verify such protocols as formal verification helps in identifying the violated properties as well as in reformulating such protocols so that they are more robust.

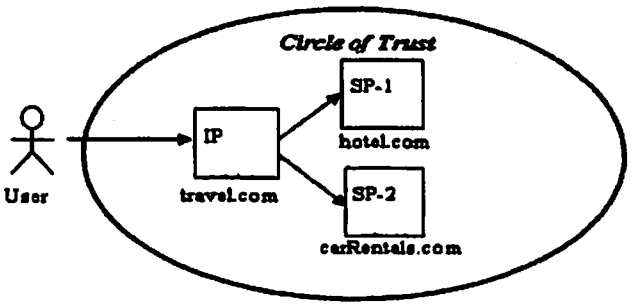


Fig. 1. Single Sign-on within a Circle of Trust

The FIM protocol is reported in Section 2. Section 3 reports our analysis of attacks and finally the concluding remarks and future work is given in Section 4.

2 The FIM Protocol

Typically the roles involved in a FIM protocol are user (U), identity provider (IP) and a service provider (SP). Identities of users are to be federated so that users can be authenticated to SPs through certificated statements issued by IPs. According to established and standardised FIM protocol specifications, IP is responsible to ensure the security of the tokens issued to SP through the requester by digitally signing them to provide token authentication and integrity [2].

In [3], the following FIM protocol has been modelled using crypto CSS, and verified using a model checker PaMoChSa for the possibility of a man in the middle attack.

1. $U \longrightarrow IP : r$
2. $IP \longrightarrow S : \{r, SAML\,assertion\}_{IP^-}$
3. $S \longrightarrow U : \{ok/ko\}_{S^-}$

As described in [3], a user (U) sends his request for federation to service provider (S) which is intercepted by an identity provider (IP). The identity provider forwards a signed message, consisting of user's request along with encrypted SAML assertion to S. A SAML assertion [5] is a token identifying user, authentication statement saying that user was authenticated, and attribute list which has attributes related to user and a nonce to avoid replay attacks. S on verification of this signed message grants or denies access to the user in the form of a message signed by him. We formalise the protocol in Cryptographic Interaction Pattern calculus (cIP) and the authentication property is specified in Protocol Logic

(PL) introduced in [1]. The formalised protocol and specification of properties are verified using an automatic tool ASPASyA [4]. ASPASyA is a symbolic model checker, it uses symbolic semantics technique to address the problem of state explosion occurring when all possible combinations of inputs are considered. The cIP representation of the FIM protocol is as follows:

$$U \triangleq (ip, s)\,[out(s).in(\{?y\}_{s^+})]$$

$$IP \triangleq (ik, u, sp)\,[in(sp).out(\{sp, \{u, IP, n\}_{ik}\}_{IP^-})]$$

$$S \triangleq (ip, u, sk)\,[in(\{S, \{u, ip, ?w\}_{sk}\}_{ip^+}).out(\{ok\}_{S^-}) \,||\, in(\{S, \{u, ip, ?w\}_{sk}\}_{ip^+}).out(\{ko\}_{S^-})]$$

Each process is represented by a list of open variables followed by the actions performed by the process. Open variables are meant for already shared secrets between principals, while the actions performed by the principal can be $in(M)$ and $out(M)$. The input action $in(M)$ signifies the pattern of expected input message by the process while $out(M)$ symbolises a message which is ready to be sent on a public channel.

The user U has two open variables for sharing the names of identity provider and service provider. The action $out(s)$ performed by U says that user sends his request (message) which in this case is the name of the service provider represented by the open variable s on a public channel. In the input action $in(\{?y\}_{s^+})$, U expects a cryptogram encrypted by private key of the service provider. A message matching this input consists of data signed by the service provider, namely a cryptogram obtained using the service provider's private key, say $\{M\}_{S^-}$. If such a message arrives, y will be assigned with M.

The identity provider IP has three open variables for sharing the name of the user, the name of the service provider and secret key shared between him and service provider. IP receives the request (message) sent by user and the communication can only proceed further if s and sp correspond to same service provider. The out action $out(\{\{sp, \{u, IP, n\}_{ik}\}_{IP^-})$ of IP emits a cryptogram encrypted by private key of the IP. This cryptogram contains user's request and SAML assertion encrypted by secret key shared between service provider and identity provider. A SAML assertion as explained earlier contains user's identity, identity provider's identity and a nonce to avoid replay attacks. It is worthy to mention here that the other information in SAML assertion (like attribute list) is not important from verification point of view, and has been abstracted away in formalism.

Service provider S has open variables for sharing names of user, identity provider and the symmetric key shared between him and IP. The behaviour of service provider S is modelled using parallel composition operator ($||$) characterising the capability of S to handle two requests. In each request it expects to receive a cryptogram $(\{\{S, \{u, IP, ?w\}_{sk}\}_{IP^+})$ send by IP, the communication between two process can again take place only if the corresponding patterns match and proper decryption keys are applied. Finally S sends either grant access message $(ok)_{S^-}$ or deny access message $(ko)_{S^-}$ to the user.

The next step in verification is to specify the security property to be checked. We are interested in authentication which can be achieved by ensuring the correct exchange of nonces between correctly connected principals. Authentication can be introduced in PL syntax as below.

$$\psi 1 \triangleq \forall u : U. \forall s : S. (s_u = S_s) \rightarrow (((y_u = ok_s) \vee (y_u = ko_s)) \rightarrow \exists i : IP. (ip_u = IP_i \wedge u_i = U_u \wedge w_s = n_i)))$$

The formula states that for all instances of user and service provider, a user is connected to a service provider which implies that message (ok/ko) received by the user is the one sent by the service provider, moreover there exists an identity provider such that if user and identity provider are properly connected then the nonce received by the service provider is the one sent by the identity provider.

3 Analysis of the attacks

ASPASyA allows the checker to verify the protocol under different contexts and using different sets of intruder knowledge. Intruder knowledge is denoted by κ and it records all the messages that has been communicated between the principals during the execution of a protocol. In [1] the Dolev-Yao intruder model has been formalised by the relation $\kappa \rhd M$ representing the ability of the intruder to derive message M from κ. We omit the formal definitions that can be found in [1]. ASPASyA also provides the facility of specifying the joining formulae in order to reduce the state space. Joining formulae constrain the open variables thus allowing the user to consider particular cases of interest for verification instead of considering all possible combinations of inputs. This can be done without modifying the protocol or property specification as observed in [1].

We define a join formula below which specifies that there is at least one user, one identity provider, and one service provider.

$$\phi \triangleq \exists u : U. \exists i : IP. \exists s : S. ((s_u = S_s) \wedge (ip_u = IP_i) \wedge (ip_s = IP_i) \wedge (sp_i = S_s)) \wedge (\forall i : IP. \forall s : S. ip_s = IP_i \rightarrow sk_s = ik_i)$$

The formula says that there is an instance u of a user U, an instance i of an identity provider and an instance s of a service provider such that u is connected to i and s ($s_u = S_s \wedge ip_u = IP_i$) and, i is in turn connected to s ($ip_s = IP_i \wedge sp_i = S_s$). Moreover whenever a service and an identity provider are connected, they share a symmetric key ($ip_s = IP_i \rightarrow sk_s = ik_i$).

Further the verifier can specify the number of principals to be used in a run of the protocol. We have performed the verification of the FIM protocol using 2 and 3 principals. The results are consistent with the analysis done in [3] and no attack was found. However, during the analysis another problem in the protocol was diagnosed, which is easy to understand if one considers multi-session runs of the protocol. Consider the scenario of Fig.2, where we have two users each having

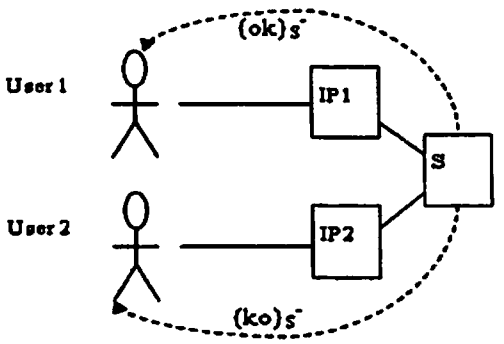


Fig. 2. A multi session scenario in a FIM protocol

a different IP and both IP are connected to same service provider. Consider the following property:

$$\psi 2 \triangleq \forall u : U.\ \forall v : U.\ y_u = y_v \rightarrow U_u = U_v$$

The formula states that if two users receive the same message they must be the same user and this property should be valid when service provider is invoked twice. However, an intruder can use his knowledge to send same message to both users, thus making the protocol behave incorrectly. The attack can be reported as under:

1. $U_1 \rightarrow IP_1 : S \quad \kappa \rhd S$
2. $IP_1 \rightarrow S : \{S, \{U_1, IP_1, W_1\}_{k_1}\}_{IP_1^-} \quad \kappa \rhd \{S, \{U_1, IP_1, W_1\}_{k_1}\}_{IP_1^-}$
3. $S \rightarrow U_1 : \{ok\}_{S^-} \quad \kappa \rhd \{ok\}_{S^-}$
4. $U_2 \rightarrow IP_2 : S \quad \kappa \rhd S$
5. $IP_2 \rightarrow S : \{S, \{U_2, IP_2, W_2\}_{k_2}\}_{IP_2^-} \quad \kappa \rhd \{S, \{U_2, IP_2, W_2\}_{k_2}\}_{IP_2^-}$
6. $I \rightarrow U_2 : \{ok\}_{S^-} \quad \kappa \rhd \{ok\}_{S^-}$

The intruder is able to generate the message $\{ok\}_{S^-}$ from his knowledge and send it to U_2 (step.6). In this way it can be easily seen that U_1 and U_2, both are receiving same message thus violating the property that if two users receive the same message they should be the same user. The attack was possible since the users have no way to check that message received by them is the one intended for them, therefore intruder can easily forward the same message to two different users. In order to correct the protocol behaviour following modification in the

protocol is recommended.

$$U \triangleq (ip, s)\,[out(s, nu).in(\{?y, nu\}_{s+})]$$
$$IP \triangleq (ik, u, sp)\,[in(sp, ?nu).out(\{sp, nu, \{u, IP, n\}_{ik}\}_{IP-})]$$
$$S \triangleq (ip, u, sk)\,[in(\{S, ?nu, \{u, ip, ?w\}_{sk}\}_{ip+}).out(\{ok, nu\}_{S-}) \,||$$
$$in(\{S, ?nu, \{u, ip, ?w\}_{sk}\}_{ip+}).out(\{ko, nu\}_{S-})]$$

In this variation of the above protocol a nonce has been added in the service provider's response. This nonce will be the same generated by the user, thus enabling the user to verify that message received by him is the one for him and not a forged message.

4 Conclusion

FIM provides an infrastructure for sharing user's authentication and authorisation information with the organisations operating in a CoT, and its fundamental goal is to enable *Single Sign-on* for providing seamless access to the services exposed by other members in that CoT. In future, we intend to extend the FIM approaches to define service authentication mechanism for sharing services' authentication information dynamically while services interact with each other in a secure SOC environment.

The analysis given in the paper shows that protocol can withstand a man in the middle attack as reported in [3] but it is not completely error free. We have suggested an amendment in the protocol to make it behave correctly. In future, we will try to find some efficient ways of reducing the state space by using heuristic techniques. One of the ways to achieve this can be use of different join formulae and property specifications in protocol logic, which force ASPASyA [4] to find states where property is violated without going through irrelevant states.

References

1. Andrea Bracciali, Gianluigi Ferrari, Emilio Tuosto, "A Symbolic framework for multi-faceted security protocol analysis", In International Journal of Information Security, Volume 7, pages 55-84, 2008.
2. Jan Camenisch, Thomas Gross, Dieter Sommer, "Enhancing Privacy of Federated Identity Management Protocols: Anonymous Credentials in WS-Security", In Proc. The 5th ACM workshop on Privacy in electronic society, pages 67-72, 2006.
3. Maurice ter Beek, Corrado Moiso, Marinella Petrocchi, "Towards Security Analyses of an Identity Federation Protocol for Web Services in Convergent Networks", In Proc. Third Advanced International Conference on Telecommunications (AICT'07), 2007.
4. G. Baldi, A. Bracciali, G. Ferrari, and E. Tuosto, "ASPASYA: Automated tool for Security Protocols Analysis based on Symbolic Approach", Available at http://www.cs.le.ac.uk/people/et52/aspasya/aspasya.html

5. Liberty Alliance ID-FF 1.2 Specifications, Available at http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications
6. Security Assertion Markup Language (SAML) V2.0 Technical Overview, Working Draft 10, 9 October 2006, Available at http://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf
7. Web Services Federation Language (WS-Federation), Version 1.1, Decemeber 2006, Available at http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP
8. Latifa Boursas, "Virtualization of the Circle of Trust amongst Identity Federations", In Proc. 1st International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and New Technologies, October 2007.
9. Bonifati, M., De Lutiis, P., Moiso, C., Morello, E., Sarchi, L.," Identity Federation for Services in Convergent Networks", In Proc. International Conference on Intelligence in service delivery Networks (ICIN 06), pages 109-114, 2006.

Towards Reliable Web Service Discovery through Behavioural Verification and Validation

Ervin Ramollari[1], Dimitrios Kourtesis[1], Dimitris Dranidis[2], and Anthony J. H. Simons[3]

[1] South East European Research Centre (SEERC),
Research Centre of the University of Sheffield and CITY College
17 Mitropoleos Str., 54624 Thessaloniki, Greece
{erramollari,dkourtesis}@seerc.org
[2] Computer Science Department, CITY College,
Affiliated Institution of the University of Sheffield
Tsimiski 13, 54624 Thessaloniki, Greece
dranidis@city.academic.gr
[3] Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
a.simons@dcs.shef.ac.uk

Abstract. Currently, the issues of trust and dependability on third-party Web services have become key challenges to the success of Service Oriented Computing in industrial environments. An important obstacle in dealing with these issues is that the current WSDL standard lacks support for explicating a service's behavioural semantics, and this can lead service consumers into binding to services with undesirable behaviour. In this paper we address this challenge by proposing an approach for reliable Web service discovery through behavioural verification and validation based on stream X-machines (SXMs). SXMs are constructed by service providers and subsequently utilised by service brokers for model-based testing and by service consumers for service selection. The end goal is to ensure that service consumers bind to Web services providing a suitable behaviour, and a verified implementation.

Key words: behavioural verification and validation, formal modelling, Web services, SOA, stream X-machines

1 Introduction

Service Oriented Computing (SOC) is an emerging paradigm for developing distributed applications using powerful abstractions called *services*. The vision of SOC is to bridge the gap between the business and IT domains, to promote business agility, and to facilitate integration of heterogeneous systems within and across organizational boundaries. Currently, the most promising implementation alternative for SOA is the Web services technology stack, founded on widely accepted standards such as WSDL, SOAP, and UDDI.

With the increasing popularity and adoption of Web services, the issues of trust and dependability on third-party providers are becoming increasingly important. Consumers need to ensure that the advertised Web services satisfy their requests in multiple aspects, before being integrated into their systems. However, one major obstacle in achieving these goals is that the current standard for Web service description (WSDL) lacks support for capturing the semantics relating to functional and non-functional aspects of a service. As a result, it is not possible to guarantee that a discovered service advertisement matches a service request in all respects, and this may lead to inappropriate bindings.

The approach that we put forward in this paper deals with modelling the behaviour of Web services in order to assist in the validation of service advertisements against consumer needs at the time of service selection. We propose the use of stream X-machines (SXMs) [14] as a powerful and intuitive formalism that not only allows for modelling the behaviour of a stateful Web service, but also enables the automated generation of test cases to verify that the actual service implementation conforms to that model. The approach involves all three main actors in a SOA environment, i.e. the service provider, the service broker, and the service consumer. The provider's role is to create a SXM model reflecting the behaviour of the provided Web service implementation, and attach it to WSDL during the publication process. Based on this model, the broker is able to derive the necessary test cases, which are run in order to verify behavioural equivalence between the advertised model and the implementation. Only services with successful test results are accepted in the registry. On the other hand, during the discovery process, the consumer is provided with a number of service candidates fulfilling the request. Through the provided SXM models, the consumer can validate the behaviour of candidate services against consumer needs, a process that aids in service selection. Therefore, this approach ensures that consumers bind with Web services providing a suitable behaviour, and a verified implementation.

The rest of this paper is structured as follows. *Section 2* presents a summary of related work. *Section 3* gives a description of the proposed approach from the perspectives of the service provider, the service broker, and the service consumer. In the end, *section 4* concludes the paper by suggesting directions for further work.

2 Related Work

Some existing work addresses the verification of Web services through model-based testing. *Sinha and Paradkar* [16] propose an algorithm, which translates Web service descriptions annotated in WSDL-S into an equivalent Extended Finite State Machine representation. The EFSM model is then exploited to generate an effective set of test cases to test the behaviour of the Web service. *Keum et al* [12] also use Extended Finite State Machines to model the behaviour of stateful Web services and to help derive a test set. Finite State Machines are extended with memory, as well as with computing blocks and predicate conditions

for state transitions. The results of the employed testing algorithm represent sequences of invocations of Web service operations. The authors provide experimental results showing that their method has the potential to find more faults compared to other methods, but notably, with a resulting test case set that is much larger and takes more time to execute.

Some other works have proposed the application of model-based testing of Web services in the context of more complete approaches. *Bertolino et al* [1] provide a framework where the provider augments the WSDL document with behavioural descriptions in a UML 2.0 Protocol State Machine (PSM) diagram, which is then translated to a Symbolic Transition System (STS). On the other hand, the broker utilises the provided STS model to generate a set of test cases, which are run on the Web service under test for behavioural conformance verification. Upon successful test results the Web service is published in the UDDI registry as a certified service. For this reason, the authors call their approach an "Audition" framework, where the Web service undergoes a monitored trial before being put "to stage".

Heckel and Mariani [7] describe a reliable Web service discovery approach, in which both the behaviour the provided service and the consumer's request are modelled using graph transformation rules. A test case derivation method is employed to test whether the service implementation conforms to the provided model. This verication is performed by the service broker before services are accepted in the registry, resulting in what the authors refer to as high-quality service discovery agencies. In addition, the broker enables matchmaking of request and advertisement models that are expressed as graph transformation rules, in order to return service candidates satisfying the consumer's behavioural constraints.

One strength of the approach described in this paper, relative to the existing ones, is that the employed X-machine functional testing algorithm is proven to generate a complete set of test cases that can reveal all faults in the implementation under test [8,9]. In addition, this approach does not require the consumer to specify a formal model of the service request during discovery, since service selection is performed through behavioural validation at the consumer site. Indeed, it is impractical to assume that the consumer knows in advance the detailed behaviour of the requested Web service and can create a formal model of that behaviour.

3 Description of the Approach

The approach that we propose in this paper, as illustrated in Figure 1, involves all the three main participants in a SOA environment, i.e. the service provider, the service broker, and the service consumer. It is based on formally modelling behavioural aspects of Web services as stream X-machines, aiming to achieve formal verification and validation of Web services during the publication and discovery process. With behaviour of a service we refer to the interaction protocol that is assumed when the operations of that service are invoked in sequence, as

well as the preconditions and effects of each operation. Behaviour is particularly relevant in stateful Web services, where responses of operations depend not only on the consumer input, but also on the internal state of the Web service.

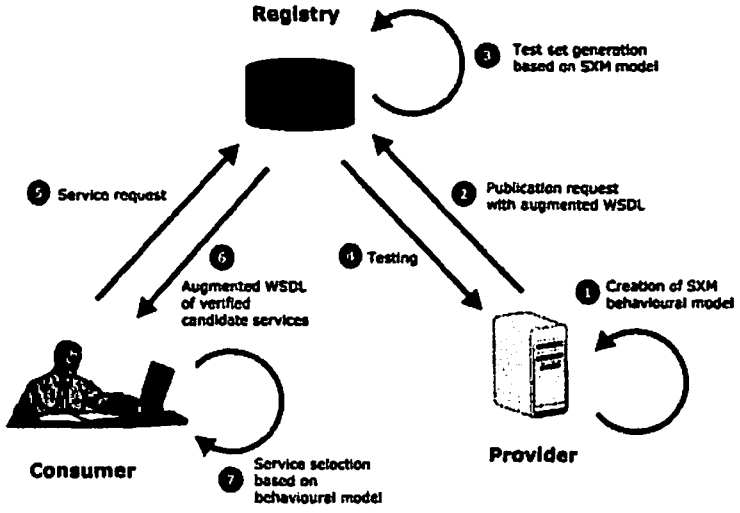


Fig. 1. Web service behavioural verification and validation approach.

The role of each participant in the scheme is fulfilled through a number of steps, as described in the following subsections.

3.1 Provider's Perspective

The service provider goes through data-level and behavioural-level analysis to derive a formal model reflecting the behaviour of the Web service that is to be published, using the stream X-machine (SXM) formalism [14, 8]. Stream X-machines (SXMs) [14, 8] are a computational model capable of modelling both the data and the control of a system. SXMs are special instances of X-machines introduced by Samuel Eilenberg in 1974 [4], which also have input and output streams. They employ a diagrammatic approach of modelling the control by extending the expressive power of finite state machines. In contrast to finite state machines, SXMs are capable of modelling non-trivial data structures by employing a memory, which is attached to the state machine. Additionally, transitions between states are not labelled with simple input symbols but with processing functions. Processing functions receive input symbols and read memory values, and produce output symbols while modifying memory values. The reader can refer to our previous work in [3] for a more detailed description of the Web service modelling procedure, using a shopping cart scenario.

The SXM model, expressed in a markup language such as XMDL [11], is then linked to the WSDL document of the Web service. Practically, this may be achieved by adding an SAWSDL annotation [6] that references the URI of the SXM markup document. The next step by the provider is the publication of the Web service to a service registry maintained by a broker. The publication query, which references the semantically annotated WSDL document at the provider site, initiates the publication procedure at the broker site.

3.2 Broker's Perspective

In this approach, an important role of the service broker is to verify the behaviour of the provided Web service implementation through model-based testing, and upon successful test results, to accept it in the service registry. This step is necessary to ensure that the implementation of the provided Web service really conforms to the advertised behavioural specifications. It is possible that this might not be the case, either because of insufficient testing at the provider site, or because of malicious intent.

With the attached SXM specification, the broker is able to derive the test sequences for verification automatically. The theory of complete functional testing from X-machines offers a method for deriving a complete, finite set of test cases, which is proven to find all faults in the implementation under test [9, 8]. Examples of faults that can be detected in the implementation include erroneous transition labels, erroneous next-states, missing states, extra states, etc [10]. The testing method is a generalization of the W-method [2]. It works on the assumption that the system specification and the implementation can be both represented as stream X-machines with the same type Φ (i.e. both specification and implementation have the same processing functions) and Φ satisfies the following design for test conditions: completeness with respect to memory (all processing functions can be exercised from any memory value using appropriate inputs) and output distinguishability (any two different processing functions will produce different outputs if applied on the same memory/input pair). Our previous work in [3] illustrates the test case derivation method using a shopping cart scenario.

The executable tests are then run by a testing engine that communicates with the Web service via SOAP messages. If the test results are successful, i.e. the expected and produced outputs match, then the Web service implementation has been shown to be free of faults with respect to the behavioural specifications. If this is the case, an advertisement of the Web service is created and added to the service registry. The benefit of performing this procedure at the broker site, as opposed to performing it at the consumer site upon discovery, is that it needs to be done only once. Since only successfully tested Web services are accepted by the broker, consumers are ensured that the Web services they discover have been verified with respect to their specifications.

3.3 Consumer's Perspective

As a first step during discovery, the service consumer formulates a service request and submits it to the service registry. In response, the service broker returns a set of annotated service descriptions that match the service request. Notably, our approach is not bound to any particular matchmaking mechanism, so that any existing mechanism may be employed to perform syntactic or semantic matchmaking between the service request and the service advertisements.

The service consumer can take advantage of the SXM behavioural model provided with each service candidate, in order to perform service selection. This is a validation process where the consumer ensures that a service model satisfies his or her requirements. An important validation technique is *model animation*, during which the user feeds the model with sample inputs and observes the current state, transitions, processing functions, memory values, and last but not least, the outputs. For example, X-System is a prolog-based tool supporting the animation of stream X-machine models [11]. In addition, *model checking* may be employed on the SXM model to check for desirable or undesirable properties, which are specified in a temporal logic formula. Research on X-machines offers a model-checking logic, called XmCTL, which extends Computation Tree Logic (CTL) with memory quantifiers in order to facilitate model-checking of X-machine models [5]. Alternatively, if the consumer has a SXM model of the required service, it can be validated by state and transition refinement against the published SXM of the provided service [15].

4 Conclusions and Further Work

The approach described in this paper is supported in fragments by a number of existing tools, which have been developed during previous research. However, several gaps exist in the required supporting infrastructure, and future research will address the consolidation of techniques and tools into a comprehensive application framework with industrial applicability. The main focus will be on the broker infrastructure, which requires more substantial work to support fully-automated Web service testing and, possibly, behavioural matchmaking. We have already developed a semantically-enhanced, UDDI-based service registry supporting the SAWSDL specification, as part of the EU-funded STREP project FUSION [13]. In order to support automated Web service verification, we are planning to integrate the semantic registry with tools for test case generation from XMDL specifications, and with capabilities for runtime testing of a Web service implementation. Additionally, in order to support behavioural matchmaking, we are planning to define an abstract query language for the service consumer and extend the current matchmaking algorithm of the semantic service registry to match the behavioural query with the advertised SXM models.

References

1. A. Bertolino, I. Frantzen, A. Polini, and J. Tretmans. Audition of web services for testing conformance to open specified protocols. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, pages 1–25, 2006. LNCS 3938.
2. T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.
3. D. Dranidis, D. Kourtesis, and E. Ramollari. Formal verification of web service conformance through testing. *Annals of Mathematics, Computing & Teleinformatics (AMCT)*, 1(5):36–43, 2007.
4. S. Eilenberg. Automata, languages and machines. *Academic Press, New York*, A, 1974.
5. G. Eleftherakis, P. Kefalas, and A. Sotiriadou. Xmctl: Extending temporal logic to facilitate formal verification of x-machines. *Matematica-Informatica*, 50:79–95, 2002.
6. J. Farrell and H. Lausen. *Semantic Annotations for WSDL and XML Schema*. W3C Candidate Reccomendation, January 2007. Available at: http://www.w3.org/TR/sawsdl/.
7. R. Heckel and L. Mariani. Automatic conformance testing of web services. In *FASE 2005*, pages 34–48. Springer, 2005.
8. M. Holcombe and F. Ipate. *Correct Systems: Building Business Process Solutions*. Springer Verlag, Berlin, 1998.
9. F. Ipate and M. Holcombe. An integration testing method that is proven to find all faults. *International Journal of Computer Mathematics*, 63:159–178, 1997.
10. F. Ipate and R. Lefticaru. State-based testing is functional testing. *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, pages 55–66, 10-14 Sept. 2007.
11. E. Kapeti and P. Kefalas. A design language and tool for X-machine specification. In *Proceedings of the 7th Panhellenic Conference on Information Techology, Greek Computer Society, Ioannina*, 1999.
12. C. Keum, S. Kang, and I. Y. Ko. Generating test cases for web services using extended finite state machine. In *TestCom 2006*, pages 103–117. Springer, 2006.
13. D. Kourtesis and I. Paraskakis. Combining sawsdl, owl-dl and uddi for semantically enhanced web service discovery. In S. Bechhofer et al., editor, *ESWC 2008*, page 614628. Springer-Verlag Berlin Heidelberg, 2008. LNCS 5021.
14. G. Laycock. *The Theory and Practice of Specification-Based Software Testing*. PhD thesis, Dept of Computer Science, Sheffield University, UK, 1993.
15. A. J. H. Simons. A theory of regression testing for behaviourally compatible object types. *Software Testing, Verification, and Reliability*, 16(3):133–156, August 2006.
16. A. Sinha and A. Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *TAV-WEB'06*, pages 17–22, Portland, Maine, USA, 2006. ACM.

Semantic Web Service Offer Discovery with Lightweight Semantic Descriptions*

Jacek Kopecký and Elena Simperl

Semantic Technology Institute (STI Innsbruck)
Innsbruck, Austria
(firstname.lastname)@sti2.at

Abstract. Semantic Web Services (SWS) are a research effort aimed at automation of the usage of Web services, a necessary component for the Semantic Web. Offer discovery is an important part of the general discovery process of finding the most suitable services for a user's goal. Nevertheless, the task of offer discovery has been largely ignored by Semantic Web Services frameworks. In this paper, we present a solution for offer discovery that uses WSMO-Lite, the new lightweight semantic Web service annotation framework.

1 Introduction

The Semantic Web is not only an extension of the current Web with semantic descriptions of data; it also needs to integrate services (e.g. e-shops and hotel reservations) that can be used automatically by the computer on behalf of its user. A major technology for publishing services on the Web is the so-called *Web services*. Based on WWW standards HTTP and XML, Web services are gaining significant adoption in areas of application integration, wide-scale distributed computing, and business-to-business cooperation. Still, many tasks commonly performed in service-oriented systems remain manual (performed by a human operator), and services in areas such as business-to-customer (e-commerce) mostly remain available only through a human interface (HTML).

In order to make Web services part of the Semantic Web, the research area of Semantic Web Services (SWS) investigates ways to increase the level of automation around Web services. Typical tasks automated by SWS technologies are discovering available services and composing them to provide more complex functionalities.

SWS automation is supported by machine-processible semantic descriptions that capture the important aspects of the meaning of service operations and messages. SWS descriptions are processed by a semantic execution environment (SEE, for instance WSMX [2]). A user can submit a concrete *goal* to the SEE, which then accomplishes it by finding and using the appropriate available Web services. SWS research focuses mainly on how the SEE "finds the appropriate Web service(s)", as illustrated in Figure 1 with the first four SEE tasks.

In the figure, the user wants to arrange a June vacation in Rome. There are four services with published descriptions: the airline Lufthansa, and hotel reservation services for New York, Rome, and one for the Marriott chain worldwide. The SEE first *discovers*

* This work is partially funded by the EU research project SOA4All.

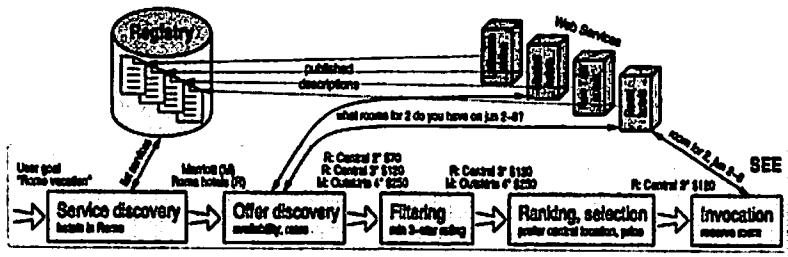


Fig. 1. Semantic Execution Environment (SEE) automation tasks

services that may have hotels in Rome, discarding Lufthansa which does not provide hotels, and the New York service which does not cover Rome. Then the SEE *discovers offers* by interaction with the discovered services. The available offers are a 4* Marriott at the outskirts of Rome, and one 2* and one 3* hotel in the city center. Then the SEE *filters* the offers depending on the user's constraints and requirements (minimum 3-star rating), *ranks* them according to the user's preferences (central location, then price) and *selects* one offer, in the end *invoking* the respective service.

While this example comes from e-commerce, the process of discovery, filtering, ranking and invocation applies in general for any SWS deployment. In some cases, when the semantic description of a service is not sufficient for the SEE to determine whether a service will satisfy the user's goal, discovery is split into service discovery, which finds services that can *potentially* fulfill the goal, and offer discovery, which interacts with the discovered services in order to find out any concrete offers. In [3], we define and motivate offer discovery in more detail.

A typical case where offer discovery is necessary is e-commerce: it would be impractical (or worse) to attempt to describe an e-commerce service completely, as it would require the description to contain an up-to-date catalogue together with availability and delivery information. Instead, the service is described as e-commerce, selling products in certain categories (e.g. hotel or flight reservations in our case above) and perhaps with other information (such as location scope for the New York hotel service); the product catalogue and availability is reachable at runtime through an invocation interface provided by the service.

There are a number of SWS frameworks that support SWS automation; the major two are WSMO [6] and OWL-S [7]. Both are built from the top down, providing semantic descriptions tailored for the expected automation tasks. However, neither of these frameworks supports the particular task of offer discovery; they have to be extended with further constructs, such as a "data-fetching interface" introduced in [9].

Recently, we have proposed WSMO-Lite [8] as a lightweight SWS framework that adds semantic annotations as a layer on top of established Web service technologies. It is built from the bottom up and it provides semantic annotations for WSDL[1], based in a clean model of service semantics, instead of tailoring the semantic constructs towards any particular tasks. As a consequence, WSMO-Lite annotations are more reusable and

[1] Web Service Description Language, `http://w3.org/TR/wsdl20`

they make it easier to realize various SWS automation functionalities. In this paper, we discuss a concrete realization of offer discovery which uses WSMO-Lite semantic descriptions.

This paper is structured as follows: in Section 2, we define offer discovery and in Section 3, we discuss the necessary semantic annotations. Section 4 provides details of our implemented offer discovery algorithm. In Section 5, we discuss some related work, and Section 6 concludes the paper.

2 Web service offer discovery

Within the big picture of SWS automation, offer discovery follows the task of service discovery, and its results go into filtering, ranking and selection. Service discovery returns a set of services that can potentially fulfill the user's goal. Offer discovery interacts with these services (or the service providers) and finds out any concrete offers that are relevant to the goal; the result of offer discovery is the set of available offers. This set is then subject to filtering and ranking according to the user's constraints and preferences. In the end, the best offer is selected and *consumed*, i.e. the client invokes the service that gave this offer and, after successful invocation, it will get the offered product or functionality.

In order to be able to talk about offer discovery, we need to specify what we mean by the term "offer". An offer O is a tuple that contains two sets of parameters (data values or types): execution parameters P_x that are required for invoking the service and consuming this offer, and extra parameters P_e that help the client to filter and to rank the offers:

$$O = \langle P_x, P_e \rangle$$

All the parameters (both P_x and P_e) can be used for filtering and ranking of offers; the extra parameters are distinguished from the execution parameters because they are not necessary for consuming the offer. The distinction affects the creation and handling of offers, discussed further on in this paper.

In our hotel scenario, the execution parameters are the start and end dates of the stay, the number and personal data of the guests; and of course the selected hotel reservation service and the concrete hotel. All this data is necessary for making a reservation. The extra parameters for filtering and ranking would be the locations and ratings of the hotels and the room prices. Strictly speaking, the client does not need to know the price when reserving a room, and it certainly does not need to send the price to the service.

Some of the execution parameter values come from the user goal, in our case the data of the guests and the dates of the stay. Further, the particular service that supplies any given offer is itself an execution parameter of the offer. This ensures that the offer is a self-contained construct for the further SWS automation steps: the invocation component knows what service to invoke; and in filtering and ranking the client may express constraints or preferences directly on the services, for instance by building trust with particular providers that delivered good value in the past. Any other offer parameter values come from the offer-discovery interaction with the services.

The split of offer parameters into execution parameters and extra ones allows us to establish identity for offers — two offers are equivalent iff their execution parameter sets are the same:

$$O_1 = \langle P_x^1, P_e^1 \rangle,\ O_2 = \langle P_x^2, P_e^2 \rangle : \quad O_1 \equiv O_2 \ \Leftrightarrow\ P_x^1 = P_x^2 \tag{1}$$

This equivalence relation comes from the fact that only the execution parameters are used in the invocation phase, when an offer is consumed. If two offers vary only in the extra parameters, the service cannot know which of them the client intends to consume. Hence, the offer discovery process must assure that it does not produce different but equivalent offers.

Finally, offer discovery (function *DiscO*) maps a set of discovered Web services $\{S_i\}$ into a set of non-equivalent offers $\{O_j\}$ from these services. Below, we formalize that no pair of offers produced by offer discovery must be equivalent (Eq. 2), and that every offer contains a parameter that ties it to the service that provides it (Eq. 3):

$$DiscO(\{S_i\}) \rightarrow \{O_j\}$$

$$\forall o_1, o_2 \in \{O_j\} : o_1 \equiv o_2 \Leftrightarrow o_1 = o_2 \tag{2}$$

$$\forall o \in \{O_j\},\ o = \langle P_x, P_e \rangle : \exists! s \in \{S_i\} : s \in P_x \tag{3}$$

Initially, we simplify offer discovery to deal with a single discovered service at a time, but it is possible that a more sophisticated offer discovery implementation can negotiate with multiple services in parallel, and pitch them one against another in order to get better deals; for example, a retailer can promise to match any competitor's price, so the offer discovery process would need to get the competitors' offers first and then use them to get matching counteroffers from the retailer.

Similarly, it is possible that the input to offer discovery for some service comes from offers of another service. For example, a retailer can offer products of varying dimensions, and a delivery service can offer different prices for different sizes. Therefore, offer discovery may also be investigated in the future in context of service composition.

3 WSMO-Lite semantic annotations for offer discovery

Semantic offer discovery should optimally be able to communicate with any Web service that provides operations for finding information about its offers. To achieve this, the offer discovery engine needs a description of the service interface, to see what operations it contains that can be used to gather offer information; and a description of the exchanged data, to understand the offers and to be able to compare them against the goal. The WSMO-Lite SWS description framework [8], together with the underlying SAWSDL standard[2], provide all the necessary semantic descriptions.

Any Web service, described in WSDL, has an *interface* which consists of a number of *operations*. Web service interfaces often intermix operations for offer inquiry with operations that actually provide the resulting product or service, for instance a hotel reservation service would provide the availability inquiry operations along with

[2] Semantic Annotations for WSDL and XML Schema, `http://w3.org/TR/sawsdl`

the operations for making reservations. For the purposes of automated offer discovery, we will use operations that only provide information and do not have any significant side-effects. In other words, we need what the Web architecture [1] calls "safe interactions[3]". WSMO-Lite allows operation annotations with functionality categories, and the WSDL 2.0 defines an operation safety flag[4], which we treat as a WSMO-Lite category for safe operations.

It remains to be seen whether all safe operations can be treated as "offer-inquiry" operations, but due to their safety, there is no harm in invoking such operations even if they do not actually help get information about the service offers.

Further, WSMO-Lite annotates operation inputs and outputs with pointers to ontology entities, such as RDFS classes. These annotations allow the semantic client to match its goal data against the inputs of the offer-inquiry operations, and to match the goal and offer data against the inputs of the execution operations[5], to distinguish between execution and extra parameters.

While WSMO-Lite describes services, the modeling of user goals is outside its scope. The representation of goals depends on the concrete SEE that implements SWS automation; in our work, we use WSMX as the SEE, so goals are described in WSMO.

In summary, WSMO-Lite provides semantic service descriptions and WSMO provides goal descriptions, which are together sufficient for us to realize an automated offer discovery process, described in the following section. Alas, due to space constraints we cannot provide an example annotated WSDL document or an example user goal which would clarify what the annotations look like in practice.

4 Offer discovery algorithm and implementation

Algorithm 1 shows how we have realized offer discovery on top of the semantic annotations described above. It is an initial working algorithm that treats the discovered services independently; this is not a serious limitation, though, as the currently available public Web services do not even allow more complex multi-party negotiations.

In short, the algorithm starts by creating an initial offer from the goal data, then it keeps invoking the offer-inquiry operations for every offer that does not provide all the required execution parameters (so-called *incomplete offers*).

To get from the goal data to all the required execution parameters for each offer, we use *planning* (cf. [5]). The currently known data (from the goal and from the current offer) becomes the initial state of the planning problem, and the presence of all execution parameters is the goal state. In our current implementation, we represent the presence of a value of a certain type with a predicate EXISTS(*type*), which is also the only predicate in the planning problem. The offer-inquiry operations are the possible transitions: the precondition of an operation is that values exist for all the input types, and the effect is

[3] Information retrieval is the canonical example of a safe interaction: the client may query a service about the availability of hotel rooms, yet by issuing the query the client makes no commitment to book the room.

[4] `http://www.w3.org/TR/wsdl20-adjuncts/#safety`

[5] Execution operations are those operations that are used when invoking a Web service in order to consume an offer.

that values exist for all the output types. Note that the offer inquiry operations do not have negative effects (or *delete lists*) because we currently assume monotonic behavior where new information cannot invalidate what was known earlier.

When a plan is found for a given incomplete offer, the first operation in this plan is invoked (its inputs must be available because otherwise it could not be the first operation of the plan). The outputs of the operation are added to the offer parameters. A single offer inquiry operation may return a list of offers; for instance, `findAvailableHotels(dates,guests)` can return multiple hotels at a time. This is detected if multiple instances in the output data are of the same type as some needed execution parameter. We assume that a single parameter in one offer can only have a single value (which may be structured and complex), therefore on line 16, the initial incomplete offer is cloned as many times as necessary, each of the clones getting a single value from the received outputs of the same type.

In the hotel scenario, booking a room requires the dates and guest data (from the user goal) and a selected hotel, so the algorithm would invoke an operation such as `findAvailableHotels(dates,guests)`, and the resulting list of hotels would complete the list of offers.

In the end (line 17), the algorithm tries to gather further extra parameters for all the known offers. This is necessary because the main algorithm only satisfies the execution parameters, so it would never invoke an operation such as `getPrice(hotel,dates)`, because the resulting *price* value is not an execution parameter. We are working on heuristic approaches for selecting the operations to invoke for the extra parameters.

Algorithm 1: Offer discovery for a single service

Input: Service *S* whose offers should be discovered, user goal *G*.
Result: The set of offers provided by *S*.

```
1  set up initialOffer with execution parameter values from G, S
2  set up incompleteOffers = {initialOffer} and completeOffers = {}
3  identify offerInquiryOperations from S
4  while incompleteOffers is not empty do
5      offer = first(incompleteOffers)
6      if offer is complete (all execution parameters are present) then
7          move offer to completeOffers; continue while
8      set up knowledgeBase from offer and G
9      select neededOfferExecutionParams — execution parameters not satisfied by offer
10     plan = planOperations(
11                 initial state: knowledgeBase,
12                 goal state: neededOfferExecutionParams,
13                 operations: offerInquiryOperations)
14     if empty(plan) then eliminate offer; continue while
15     receivedValues = invoke(S, plan[1], knowledgeBase)
16     add receivedValues to offer (possibly splitting it into multiple offers)
17 gatherExtraParameters(completeOffers)
18 return completeOffers
```

The execution parameters mentioned on line 1 are the input parameters of the execution operations. Currently, we treat all non-safe operations as the execution operations, and we attempt to satisfy all their inputs. However, it is necessary to select only those execution operations that are relevant for the user goal: for instance, a hotel reservation service may have an operation `cancelReservation(reservationCode)` which is not going to be invoked when booking a room, and whose input parameter cannot be satisfied by discovering offers. Our algorithm would fail to find any offers here.

We have implemented the algorithm within the WSMX system [2] with positive initial results, but a larger evaluation experiment remains as future work. While we use WSMO goal descriptions, we do not support the full complexity of WSMO goals yet. And finally, the planning algorithm used in our implementation is not semantic (for instance, it cannot take into account any subclass relationships between the parameters and the available values); this limits the expressivity available for the ontologies used to describe the input and output data of the service operations. We are looking for suitable ways of extending the planning part with semantic reasoning.

5 Related work

While service discovery, filtering and ranking have been extensively researched, offer discovery has been seriously investigated only by two research teams other than ours, as far as we know.

Zaremba *et al.* [9, 10] talk about a so-called "contracting interface" with an explicit choreography that guides the execution of this interface. In their case, the WSMX client follows the predefined choreography to find out the concrete offers provided by a discovered Web service. The contracting interface can be likened to a prescribed protocol for offer discovery. In our work, we choose to lower the semantic description burden (we do not require a choreography description) by employing planning over available safe operations. Our current solution could not use unsafe operations, while they could be included in Zaremba's contracting interface. If this proves to be a real-world limitation, we will attempt to add other simple annotations that would mark operations as suitable for offer discovery.

Küster *et al.* [4] models service requests and advertisements as data graphs, which can be mapped structurally, and which can contain input and output variables. The interaction with services is broken down into two phases: *estimation* and *execution*; with input and output variables assigned for either of them. To guide the invocation of the estimation operations, the variables are marked with a simple numeric order in which they are to be used. From the published works, it is unclear how exactly their work ties to concrete Web services and operations, but Küster's approach seems in effect similar to Zaremba's: it requires more complex semantic descriptions than our approach, in exchange for potentially greater expressivity. However, Küster's simple numerical ordering of estimation invocations may be limiting when compared to Zaremba's more generic choreographies.

6 Conclusions

Web services are a necessary part of the Semantic Web, and research on Semantic Web Services aims to automate their use. Among the tasks that can be automated using semantic technologies is offer discovery, especially important for e-commerce applications. In this paper, we have presented a formalization of offer discovery and we have shown an offer discovery algorithm and its implementation using WSMO-Lite semantic descriptions.

While our prototype gives positive results, there are some open points in need of solutions, and we still need to perform a proper evaluation experiment for our approach.

References

1. Architecture of the World Wide Web. Recommendation, W3C, December 2004. Available at `http://www.w3.org/TR/webarch/`.
2. A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. *International Conference on Web Services (ICWS 2005)*, July 2005.
3. J. Kopecký, E. Simperl, and D. Fensel. Semantic Web Service Offer Discovery. In *Proceedings of Service Matchmaking and Resource Retrieval in the Semantic Web Workshop, colocated with 6th ISWC*, 2007.
4. U. Küster and B. König-Ries. Supporting dynamics in service descriptions — the key to automatic service usage. In *Proceedings of the Fifth International Conference on Service Oriented Computing (ICSOC07)*, Vienna, Austria, September 2007.
5. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
6. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
7. The OWL Services Coalition. OWL-S 1.1 Release. Available at `http://www.daml.org/services/owl-s/1.1/`, November 2004.
8. T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, Tenerife, Spain, 2008.
9. T. Vitvar, M. Zaremba, and M. Moran. Dynamic service discovery through meta-interactions with service providers. In E. Franconi, M. Kifer, and W. May, editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.
10. M. Zaremba, T. Vitvar, M. Moran, and T. Hasselwanter. WSMX Discovery for SWS Challenge. SWS Challenge Workshop, Athens, Georgia, USA, November 2006.

Tools4BPEL4Chor

Niels Lohmann[1] and Oliver Kopp[2]

[1] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de
[2] Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
kopp@iaas.uni-stuttgart.de

1 Tool Support for BPEL4Chor Choreographies

We present several tools support the modeling, analysis, synthesis, and correction of BPEL4Chor choreographies [1]. Fig. 1 presents an overview and the relationship between the tools.

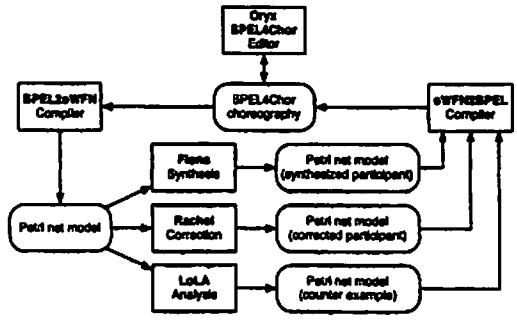


Fig. 1. Tool chains to analyze BPEL4Chor choreographies.

Oryx: Modeling Service Choreographies. Oryx is a graphical editing framework written in JavaScript developed at the Hassso-Plattner-Institute. A BPMN editor and a BPMN to BPEL4Chor transformation has been developed[3] so that a BPEL4Chor choreography can be modeled using configured BPMN [2]. That enables business users to graphically model choreographies without the need of writing XML code themselves.

BPEL2oWFN: Translating BPEL Processes into Service Models. To analyze a BPEL process or a BPEL4Chor choreography, it has to be translated into a mathematical formalism. In the Tools4BPEL framework, we use Petri nets as formal model. A feature-complete translation from BPEL processes into Petri net models [3] is implemented in the compiler BPEL2oWFN[4].

LoLA: Analyzing Service Models. Petri nets allow for diverse analysis techniques. Furthermore, a lot of research has been conducted to ease the state space explosion problem that usually makes the analysis of real-world models

[3] Available at `http://www.bpel4chor.org/editor`.
[4] Available at `http://service-technology.org/bpel2owfn`.

impossible. The model-checking tool LoLA[5] [4] implements a variety of reduction techniques which allows, for example, deadlock checking of choreographies of thousands of participants [5].

Fiona: Synthesizing Service Models. While the analysis of choreographies may help to find design flaws in the interaction between all participating services, Petri net models may also support the design of choreographies. With the tool Fiona[6] [6], missing participants can be synthesized which are guaranteed to work deadlock-freely in the choreography.

Rachel: Fixing Service Choreographies. In case a service choreography deadlocks because of one participant, one solution might be to replace that participant by a synthesized participant. Though this replacement guarantees correctness, it gives no hint how to repair the incorrect participant. Rachel[7] [7] is a tool providing such hints: It calculates the minimal edit actions necessary to change the participant to achieve deadlock-freedom.

oWFN2BPEL: Translation Service Models into BPEL Processes. To use the analysis results and the synthesized participants in the original BPEL choreography, the compiler oWFN2BPEL[8] [8] translates Petri net models into abstract BPEL processes. These processes already describe the business protocol of the BPEL process with its partners. Additionally details such as data aspects or fault handling can then be added manually to refine the process towards executability.

References

1. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, IEEE (2007) 296–303
2. Decker, G., Kopp, O., Leymann, F., Weske, M.: Modeling service choreographies using BPMN and BPEL4Chor. In: CAiSE '08, Springer (2008)
3. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 77–91
4. Schmidt, K.: LoLA: A low level analyser. In: ICATPN 2000. Volume 1825 of LNCS., Springer (2000) 465–474
5. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 46–60
6. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: BPM 2006. Volume 4102 of LNCS., Springer (2006) 17–32
7. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, Springer (2008) to appear.
8. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. Volume P-127 of LNI., GI (2008) 57–72

[5] Available at `http://service-technology.org/lola`.
[6] Available at `http://service-technology.org/fiona`.
[7] Available at `http://service-technology.org/rachel`.
[8] Available at `http://service-technology.org/owfn2bpel`.

YR-SOC 2008 is sponsored by:

Organising Committee

Monika Solanki	(Imperial College London, UK)
Barry Norton	(Open University, UK)
Stephan Reiff-Marganiec	(University of Leicester, UK)

Programme Committee

We are extremely grateful to the committee for their reviews of the submissions:

Charlie Abela	University of Malta, Malta
Marco Aiello	Rijksuniversiteit Groningen, the Netherlands
Antonio Cau	De Montfort University, UK
Dimitris Dranidis	CITY College, Greece
Howard Foster	Imperial College London, UK
Helge Janicke	De Montfort University, UK
Steve Jones	Capgemini, UK
Raman Kazhamiakin	Bruno Kessler Foundation, Trento, Italy
Nora Koch	Ludwig-Maximilians-Universitaet Muenchen, Germany
Alexander Lazovik	University of Trento, Italy
Barry Norton	KMi, Open University, UK
Carlo Montangero	University of Pisa, Italy
Arun Mukhija	University College London, UK
Stephan Reiff-Marganiec	University of Leicester, UK
Marc Richardson	British Telecom, UK
Monika Solanki	Imperial College London, UK
Emilio Tuosto	University of Leicester, UK
Steven Willmott	Universitat Politecnica de Catalunya, Spain

Preface

Service Oriented Computing (SOC) is more than just ideas related to those services: in particular it is a chance of bringing together the business/user domain and the services domain. The word 'service' encompasses web services, semantic web services, grid services and e-services.

The 3rd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2008) is a 2-day workshop aimed at PhD students, young researchers working in the industry and researchers who have completed their studies in the last few years. It followed on from the highly successful events, hosted by De Montfort University in 2005 and University of Leicester in 2007, both at Leicester UK. YR-SOC 2008 took place at Imperial College London, UK, and was organised by Monika Solanki, Barry Norton and Stephan Reiff-Marganiec.

The aim of the workshop is to build a reputable and respectable forum for young researchers with inputs from industry practitioners. The core objectives are to exchange information regarding advancements in the state of the art and practice of SOC, as well as to identify the emerging research topics and define the future trends in this domain. Contributions cover aspects such as frameworks for building SOC applications, SOC composition, orchestration and choreography, SOC modelling and design, Semantic Web, ontologies, and SOC, and SOC discovery and selection (although the call was considering further areas).

The programme included invited talks from Andrew Gordon, Microsoft Research and Wolfgang Emmerich, University College London. Additonal presentations were made by Stephen Gorton (ATX Software, Portugal) and Nicolas Gold (SOSoRNet, UK). The technical programme included a rich variety of papers from young researchers across Europe, including Netherlands, Germany, Austria and the UK.

The workshop received a total of 16 submissions, which were each reviewed by at least 3 people from a strong programme committee of international reputation. The committee decided to accept 10 papers. These proceedings include all accepted submissions, which were updated in light of the reviews given.

The workshop was organised with generous sponsorship from the following organisations:

Imperial College London
Knowledge Media Laboratory
University of Leicester
Luisa
ATX Software
Microsoft Research
SOSoRNet

June 2008 Monika Solanki, Barry Norton and Stephan Reiff-Marganiec

The 3rd European Young Researchers Workshop on Service Oriented Computing

www.yrsoc.org

12-13 June 2008

Monika Solanki
Imperial College London
m.solanki@imperial.ac.uk

Barry Norton
KMi, Open University
b.j.norton@open.ac.uk

Stephen Reiff-Marganiec
University of Leicester
srm13@le.ac.uk

Imperial College London