University of Augsburg
Prof. Dr. Hans Ulrich Buhl
Research Center
Finance & Information Management
Department of Information Systems
Engineering & Financial Management

Discussion Paper WI-190

Verification of Web Service Compositions: An Operationalization of Correctness and a Requirements Framework for Service-oriented Modeling Techniques

by

Maximilian Röglinger

appears in: Business & Information Systems Engineering 1 (2009) 6

Universität Augsburg, D-86135 Augsburg
Visitors: Universitätsstr. 12, 86159 Augsburg
Phone: +49 821 598-4801 (Fax: -4899)
www.fim-online.eu

A Requirements Framework for Service-Oriented Modeling Techniques with Respect to Web Service Verification

Author:
Maximilian Röglinger

Dipl.-Wirtsch.Inf. Maximilian Röglinger
University of Augsburg
Department of Information Systems Engineering & Financial Management
Competence Center Finance & Information Management
Universitätsstraße 16
86135 Augsburg
Germany

Tel.: +49 821 598-4113
maximilian.roeglinger@wiwi.uni-augsburg.de
http://www.wi-if.de

A Requirements Framework for Service-Oriented Modeling Techniques with Respect to Web Service Verification

Abstract:

Web service compositions (WS compositions) coordinate Web services of different enterprises. They are expected to constitute the foundation of service-oriented architectures, to improve business processes as well as to foster intra- and inter-organizational integration. Especially in inter-organizational contexts, quality of service referring to non-functional requirements and conformance to functional requirements are becoming vital properties. With WS compositions being asynchronous and distributed systems, the latter – which is also called correctness – can be shown best by verification. This paper examines how correctness has to be operationalized for WS compositions. It also examines how service-oriented modeling techniques should be shaped so that correctness can be shown by verification while WS compositions can be modeled intuitively. Correctness is analyzed from a system-theoretic viewpoint. Moreover, a requirements framework for formal foundations and modeling support is proposed. In order to show the framework's principle applicability, one example approach is analyzed with respect to the corresponding requirements.

Executive Summary:

Conformance to functional requirements, which is also known as correctness, is a prerequisite of the quality of service of Web service compositions (WS compositions). This paper operationalizes correctness for WS compositions. It also proposes a requirements framework for service-oriented modeling techniques so that correctness can be shown by verification and WS compositions can be modeled intuitively.

- Correctness splits into structural and behavioral correctness. The former requires service interfaces to match and required functionality to be available. The latter requires sequences of exchanged messages to conform to previously defined behavioral claims.
- The requirements framework refers to formal foundations and modeling support. Formal foundations require formal languages for models of WS compositions and behavioral claims. Modeling support requires a process model, adequate modeling and verification tools as well as their integration.

Keywords: *Web service compositions, Service-oriented modeling, Formal methods.*

1 Motivation and Object of Research

Web services and Web service compositions (WS compositions) are currently considered to be the most widespread possibility for implementing service-oriented architectures (Erl 2004; Newcomer and Lomow 2005). A (basic) Web service is a software system that exposes its functionality by means of a machine-processable interface consisting of several operations. It enables (asynchronous) message-based machine-to-machine interaction over a network (Booth et al. 2004). Web services are offered by internal IT departments or by external service providers. A WS composition is a Web service that realizes complex functionality by coordinating multiple (component) Web services in state-full transactions (Alonso et al. 2004, p. 141). Analogous to (basic) Web services, WS compositions expose their functionality via interfaces. In this paper, WS compositions refer to the orchestration of Web services, not to their choreography (Dostal et al. 2005, p. 226).

According to a survey of the Yankee Group, WS compositions raise high expectations (Kallus 2004). Among other things, they are expected to improve business processes and to foster the integration of existing e-business, CRM, SCM, and ERP initiatives. Already in 2004, more than 50 % of the US companies relied on Web services whereof 60 % estimated their impact on business-to-business integration (very) high. However, designing and running WS compositions is error-prone. Involved teams of software engineers and modelers usually stem from multiple companies and have different skills, experiences, or functional backgrounds. Web services are usually managed by different companies and may not have been developed for composition. Analogous to other distributed systems based on asynchronous communication, it is difficult to anticipate how WS compositions behave during execution and whether they conform to the functional requirements identified during requirements engineering. Errors, however, may violate service level agreements. This may cause losses or penalties and have negative impact on the reputation of the companies involved. Therefore, it is an important task to make sure that WS compositions conform to their functional requirements.

A possibility of reliably showing conformance to functional requirements is verification. Here, this property is also called correctness. The idea of verification dates back to Floyd (1967) and Hoare (1969). In contrast to testing, for instance, where correctness can only be shown with respect to selected test data, verification aims at exhaustively proving correctness for all behavioral facets and inputs of a given program. During verification, the program is analyzed automatically. This requires the semantics of each program statement to be defined unambiguously. A program is said to be correct if its implementation (i. e. its program code or a corresponding formal model) is consistent with its specification of behavioral claims (i. e. functional requirements on the program's behavior) (Balzert 1998, pp. 445-472).

Despite the need for correctness, research focuses on non-functional requirements such as availability, scalability, capacity, etc. (Lee et al. 2003), which are already known from network research. Current approaches to service-oriented modeling do not (or hardly) cope with correctness and verification. This includes specification languages like the de facto standard WS-BPEL (Alves et al. 2007) – despite some attempts (e. g. Farahbod et al. 2005; Moser et al. 2007; Lohmann 2007; Stahl 2005) – and high-level approaches (Arjansani 2004; Zimmerman et al. 2004; Papazoglou and van den Heuvel 2006). Current approaches to the verification of Web services (WS verification) require formal models, e. g. in terms of finite

state automata or Petri nets, which are not intuitive for conceptual modelers. Approaches to WS verification also postulate a variety of formal claims on WS compositions so that it is not clear what correctness actually means.

Against this background, we address the following research questions: *How does correctness have to be operationalized so that it fits the peculiarities of WS compositions? How do service-oriented modeling techniques have to be shaped so that correctness can be shown by verification and WS compositions can be modeled intuitively?*

This paper relies on a design-oriented, deductive, and argumentative research approach (Hevner et al. 2004, Wilde and Hess 2007). In section 2, we identify the research gap by compiling current approaches to WS verification and service-oriented modeling. Section 3 proposes a definition of correctness and a requirements framework as artifacts. Section 4 aims at showing the framework's basic applicability in the sense of a basic evaluation. Section 5 briefly summarizes the findings and points out further research.

2 State of the Art

As for WS verification, the most frequently applied verification method is model checking (Clarke et al. 2001; Schneider 2004). This is because model checking is particularly suitable for verifying distributed systems of multiple components that interact via message exchange. WS compositions are such distributed systems. In contrast to other verification methods, model checking does not directly work on implementations in the sense of program code, but on formal models that focus on relevant details such as exchanged messages and their content. A variety of model checking-based approaches has been proposed for WS verification (van Breugel and Koshkina 2006). WS compositions are usually modeled by means of finite state automata (e. g. Fu et al. 2004), Petri nets (e. g. Martens 2005; Rozinat and van der Aalst 2008; Schlingloff et al. 2005), abstract state machines (e. g. Fahland and Reisig 2005; Farahbod et al. 2004), or process algebras (e. g. Ferrara 2004). Some approaches are also capable of translating XML-based models of WS compositions (see below). Behavioral claims are commonly formalized by means of temporal logics. This is because it enables to reason about the content and temporal interdependencies of exchanged messages without introducing time explicitly (Clarke et al. 2001, p. 4).

In the following, we do not discuss the approaches in their entirety, but focus on how they deal with correctness and behavioral claims. Most approaches enable to specify behavioral claims that refer to concrete use cases. This is done by means of temporal logics as just mentioned. Some approaches additionally postulate claims that do not refer to concrete use cases. This is mostly done with respect to the formalism employed. The following list shows selected claims of the latter category:

- *Usability* requires a WS composition to terminate properly (Schlingloff et al. 2005, p. 11; Kopp et al. 2006; Martens 2005, p. 26).
- *Syntactic compatibility* requires that two Web services can be composed with respect to their interfaces, i. e. names of messages (Martens 2005, p. 23).
- *Semantic compatibility* requires that the composition of two Web services fulfills the usability claim (Martens 2005, p. 26).

- *Fitness* indicates up to which percentage the behavior of a WS composition conforms to its implementation (Rozinat and van der Aalst 2008, p. 67).
- *Appropriateness* indicates whether the implementation of a WS composition adequately characterizes the observed behavior (Rozinat and van der Aalst 2008, p. 67).

Beyond, there are also claims on the equivalence of WS compositions (Martens 2005, p. 27; Kopp et al. 2006) or on whether the communication pattern of multiple Web services could be simulated by synchronous message exchange (Fu et al. 2004, p. 627). As these claims refer to correctness at best indirectly, they are omitted for the further discussion.

The following is noteworthy: No approach states how the claims it proposes refer to the overall concept of correctness. It remains unclear whether any subset of these claims would be sufficient, whether (or how) claims of several approaches cohere, and whether they fit the peculiarities of WS compositions. Some claims depend on the underlying formalism (e. g. usability was defined for Petri nets), others are too generic (e. g. syntactic compatibility refers to the names of messages, semantic compatibility is limited to termination). It is not discussed whether these claims can be structured. Concluding, there is a research gap with respect to how correctness can be operationalized so that it fits the peculiarities of WS compositions.

When presenting an operationalization in the next section, we adopt several existing ideas in order to provide an incremental contribution. We adopt the idea that there are claims that refer to concrete use cases and others that do not. The latter category, for instance, would include usability as termination-related requirement. We extend syntactic compatibility from message names to operations and parameters. This is appropriate because correctness rather depends on the content of exchanged messages. We also extend semantic compatibility to possible (input and output) values and value ranges of parameters (see below) as well as to behavioral claims referring to concrete use cases. This enables to reason about sequences of exchanged messages and concrete behavior. We omit fitness as we consider correctness as a dichotomous property, that is, a WS composition is either correct or not. We also omit appropriateness as is takes on a contrary perspective by indicating whether an implementation characterizes observed behavior well. As contribution, we structure correctness by means of a system-theoretic perspective. This is suitable as each WS composition can be characterized by structure and behavior. In general, claims on a WS composition's structure (e. g. syntactic compatibility) do not require verification as they deal with static aspects. Behavioral claims, in contrast, require verification as they address dynamic aspects. This distinction enables to assess correctness in a less complex manner.

As for service-oriented modeling, there are technical XML-based specification languages and comprehensive high-level approaches. As regards the former category, service interfaces, operations, and parameters are formalized by means of the Web Service Description Language (WSDL) (Christensen et al. 2001). Messages exchanged to invoke other services' operations are commonly specified in terms of SOAP (Mitra 2003). WS compositions are specified by means of the Web Services Business Process Execution Language (WS-BPEL) (Alves et al. 2007), the Web Services Choreography Description Language (WSCDL) (Kavantzas et al. 2005), the Business Process Modeling Language (BPML) (Dubray 2008), or the Web Service Choreography Interface (WSCI) (Arkin et al. 2002) (for an overview see e. g. Peltz 2003). In the context of the Semantic Web, there are specification languages for modeling the semantics of operations and parameters, e. g. possible (input and output) values

and value ranges. They include the Resource Description Framework (RDF) (Klyne and Carroll 2004) and the Web Ontology Language for Web services (OWL-S) (Martin et al. 2004) (for an introduction see e. g. Herman 2003). As for high-level approaches, two elaborate examples are presented here. The service-oriented modeling and architecture approach proposes a framework for service identification, specification, and realization including role models for service providers and consumers (Arjansani 2004; Zimmerman et al. 2004). The service-oriented design and development methodology covers the entire service development lifecycle from service analysis and design to service execution and monitoring (Papazoglou and van den Heuvel 2006). It illustrates each phase in detail and proposes general design principles for SOA. It also acknowledges the importance of correctness (Papazoglou and van den Heuvel 2006, p. 435).

Both high-level approaches do not elaborate on how service-oriented modeling techniques should be shaped so that correctness can be shown by verification while WS compositions can be modeled intuitively. Moreover, the specification languages from above, e. g. WS-BPEL, are currently not or hardly amenable to verification. Thus, there also is a research gap. As the high-level approaches do not provide any concrete hint on how to integrate verification, it is not possible to make an incremental contribution. Therefore, we examined general requirements of verification and conceptual modeling with respect to WS compositions and tried to integrate them into a requirements framework. Moreover, we used the specification languages in order to define the structural requirements of correctness independent of the formalisms for WS verification.

In the following, we operationalize correctness based on the deliberations from above. We then elaborate on the requirements framework for service-oriented modeling techniques.

3 Artifacts

3.1 A Definition of Correctness for Web Service Compositions

We analyze correctness from a system-theoretic perspective. This seems appropriate because WS compositions can be interpreted as general systems characterized by structure and behavior (Ferstl and Sinz 2006, p. 12). The structure of a WS composition encompasses the WS composition itself, the component Web services, and the message types they may exchange. The latter are given by the WS composition's invocation statements and by the operations of the component Web services' interfaces. The behavior of a WS composition represents the actual interaction among the WS composition and its component Web services. This includes the set of all message sequences (i. e. sequences of operation invocations). In accordance with this point of view, we propose that correctness splits into structural and behavioral correctness.

3.1.1 Structural Correctness

WS compositions virtually have two interfaces: a "provides interface" and a "requires interface" (Sommerville 2004, p. 444). The former comprises the operations by which a WS composition provides its functionality to other Web services. The latter includes the operations a WS composition requires in order to implement its functionality. Component Web services only have a "provides interface" because they are only known from an external perspective.

Structural correctness requires that for each operation of the WS composition's "requires" interface there is at least one identically named operation in a component Web services "provides" interface that matches with respect to number, sequence, and types of parameters. One possibility to check this is to compare the component Web services' WSDL interfaces and the invocation statements of the WS composition's WS-BPEL specification. If mandatory and optional parameters are distinguished, only mandatory parameters need to match. If the WS composition and the component Web services are semantically annotated, operations also have to match with respect to possible (input and output) parameter values or value ranges. It is not necessary that each operation of the component Web services' "provides" interfaces has a matching operation in the WS composition's "requires" interface. This is because Component Web services may of course provide more functionality than required.

In many cases, component Web services are discovered and selected at run time. WS compositions may be executed although not all required operations are available. In contrast, state-of-the-art verification techniques require all operations to be available and models of WS compositions to be completely specified. Otherwise, they cannot be processed by verification tools and correctness cannot be analyzed. That is, if a component Web service is not available, one can neither reason about its own behavior nor its implications on the WS composition's global behavior. To overcome this discrepancy, verification could take place at design time or be shifted to the point in time during execution where all component services have been selected. As WS compositions and/or behavioral claims may have to be changed after verification, one would have to accept human interaction in the latter case.

3.1.2 Behavioral Correctness

Behavioral correctness requires sequences of messages to conform to a set of behavioral claims. As messages can only be exchanged if requires and provided operations match and all required operations are provided, structural correctness is a prerequisite of behavioral correctness. In literature, behavioral claims split into safety claims and liveness claims (Schneider 2004, p. 14). Safety claims are claims that must not be violated, whereas liveness claims must always hold (Holzmann 2003, p. 74). This distinction does not sufficiently characterize behavioral correctness of WS compositions. What is missing is a complementary distinction between application-independent and application-dependent claims (see section 2).

- *Application-independent claims*: Claims that cover generic issues resulting from the distributed and asynchronous nature of WS compositions are called application-independent claims. They occur in many use cases and may be captured in a rather standardized manner. There are application-independent safety and liveness claims. Typical claims of the former category are mutual exclusion and deadlock freedom. Mutual exclusion guarantees the integrity of business data (e. g. available stock, account balances) by ensuring that shared variables or other critical sections are never accessed by more than one component Web services at the same time. Deadlock freedom is necessary for the termination of WS compositions. Otherwise, it would be possible that two or more component Web services wait for one another. A typical claim of the latter category is starvation freedom, which is closely related to termination and deadlock freedom. A Web service is said to starve if it has requested a resource (e. g. a document or a database entry) that is currently being held by another Web service not willing to release it. Starvation

freedom guarantees by means of some fairness policy that each request for a resource is eventually satisfied.

- *Application-dependent claims*: Claims that vary with a WS composition's use case are called application-dependent claims. They are much more difficult to discover because they are only valid for some or only one use case. For this reason, it is impossible to enumerate them exhaustively. Some claims occur more than once and can be structured into catalogs of domain-specific claims. For example, there might be a catalog for commercial applications. A corresponding safety claim may be that customers do not have to pay for goods they have not ordered. A corresponding liveness claim may be that each customer who places an order will eventually receive an invoice. The advantage of this catalog is that it applies to all use cases where customers order goods or services. Nevertheless, some application-specific claims apply to just one scenario so that they need to be assessed individually.

Tab. 1 summarizes the types and examples of behavioral claims.

Tab. 1 Types and examples of behavioral claims

	Safety claims	**Liveness claims**
Application-independent claims	Mutual exclusion Deadlock freedom	Starvation freedom
Application-dependent Claims	"A customer never pays for goods he has not ordered."	"A customer eventually receives an invoice."

3.2 A Requirements Framework for Service-Oriented Modeling Techniques

Now that correctness has been examined, it is assessed what requirements service-oriented modeling techniques have to meet so that correctness can be shown by verification and WS compositions can be modeled intuitively. Therefore, we propose a requirements framework considering two complementary perspectives: formal foundations and modeling support.

- *Formal foundations*: Verification requires models and specifications of WS compositions to conform to formal languages and their semantics to be defined unambiguously (Balzert 1998, p. 467). This enables to "compute" all behavioral facets of WS compositions and to check which of them violate the specification. Only formally well-founded models and specifications are amenable to verification tools.

- *Modeling support*: Whereas requirements on formal foundations are compulsory with respect to technical amenability to verification, modeling techniques should also consider the modelers' capabilities and limitations of information processing. This is important for several reasons: First, in the context of business and information systems engineering as an inter-discipline, models aim at reducing complexity and at fostering the communication among modelers and model users (Ferstl and Sinz 2006, p. 123). Second, in the context of WS compositions, modelers from different enterprises with different skills, experiences, and functional backgrounds cooperate. Third, verification tools operate on a technical

level so that their output is difficult to understand for conceptual modelers. Fourth, modeling behavioral claims is error-prone so that modeling tools should support modelers as good as possible.

Tab. 2 shows the requirements framework for service-oriented modeling techniques. In the following, each requirement will be presented. The *formalization of models* of WS compositions requires formal syntax and formal semantics.

- *Formal syntax*: The syntax of a modeling language encompasses elements as well as rules that prescribe how to combine elements. To cover the behavioral facets of WS compositions, elements for manipulating the conversational state (e. g. assignment of variables), message exchange (e. g. synchronous and asynchronous send / receive), and control flow (e. g. conditions, iterations, concurrency) are necessary. Syntax is formal if it is specified in terms of non-prosaic meta models or mathematic models.

- *Formal semantics*: Semantics builds upon syntax and deals with the meaning of elements. The semantics of a WS composition represents its behavior resulting from the interplay of its elements. It should deal with issues of distributed and asynchronous systems such as concurrency and non-determinism. Most modeling languages in the field of business and information systems engineering provide formal syntax, only few provide formal semantics.

The *formalization of specifications* requires a formal language for behavioral claims, e. g. temporal logics as mentioned above. With models and specifications serving as input for the same verification tool, it is important that this formalism complies with the modeling language employed for describing the semantics.

Tab. 2 Requirements framework for service-oriented modeling techniques

Formal Foundations	**Modeling Support**

• **Formalization of models**
  • Formal syntax
  • Formal semantics
• **Formalization of specifications**
	• **Process model**
	• **Tool support**
	  • Reduction of complexity
	  • Visualization of behavior
	  • Integrated modeling of WS compositions and specifications
	  • Constructive feedback

Modeling support requires a *process model* that guides the modeler through the process of modeling and verifying WS compositions. In particular, the process model should include the modeling of specifications and their harmonization with models of WS compositions. It should contain a “loop” from verification back to the modeling of models and specifications because models and specifications may need to be modified several times after verification.

Beyond, modeling techniques should provide *modeling and verification tools* that fulfill the following requirements:

- *Reduction of complexity*: As models of WS compositions typically refer to multiple business partners and represent complex behavior, they easily overstrain the modelers' capacity of information processing. Modeling tools should provide graphical means for reducing complexity. Especially the interfaces of WS compositions enable modelers to switch between internal and external perspectives and to focus on a particular section of a model.

- *Visualization of behavior*: Behavior is often visualized in a static manner. That is, carriers of behavior (e. g. activities, functions, tasks) are identified and related according to their temporal or behavioral logic. This does not correspond to human imagination. Modeling tools should be able to simulate the execution of WS compositions. Modelers should be able to chose among different execution possibilities and get an intuitive awareness of possible errors and bottlenecks.

- *Integrated modeling of WS compositions and specifications*: Models and specifications of WS compositions cohere closely as they represent actual and expected behavior respectively. Additionally, the formalization of behavioral claims is error-prone and needs to be harmonized with the peculiarities of concrete representations of WS compositions. Modeling tools should enable the integrated modeling of WS compositions and specifications.

- *Constructive feedback*: If behavioral claims are violated, the reasons may not be immediately obvious. This is for two reasons: First, models and/or specifications may contain errors. Second, errors are difficult to reconstruct in a distributed environment. Verification tools should provide modelers with constructive feedback on where and under what circumstances behavioral claims are violated. As this feedback typically is highly technical, modeling tools should furthermore be able to represent this feedback in a way understandable for modelers.

We hypothesize that service-oriented modeling techniques that meet these requirements enable to show correctness by verification and to model WS compositions intuitively.

4 Basic Evaluation of the Requirements Framework

In order to provide a basic evaluation, we will now analyze whether the requirements framework is principally applicable. In the following, we present a selected approach, analyze it with respect to the requirements, and assess whether this leads to reasonable findings. The approach is that of Fu et al. (2004; 2004a; 2005; 2006). It has been chosen because it is an elaborate framework for analyzing, designing, and verifying WS compositions (Fu et al. 2004, p. 622), has already been applied to WS-BPEL processes, relies on the state-of-the-art model checker SPIN (Holzmann 2003), and has been published in several international journals and conferences. In order to be more illustrative, we refer to a widely known example from the previous WS-BPEL version (Andrews et al. 2003).

The example is as follows: A bank intends to acquire more business customers (BIZ). A survey disclosed that business customers complain about administrative overhead when applying for short-term loans of moderate value. The board has decided to improve the process: Future loan applications will be classified by amount and risk. The latter will be assessed by an external association of experts (ASS). Only critical applications, i. e. those

with an amount higher than or equal to 10,000 Euros or with high risk, will be examined by in-house loan approvers (APP). Business customers should apply via the Internet. As both the external experts and the internal loan approvers offer their functionality by means of Web services, the additional functionality will be implemented by means of a WS composition. Thereby, the bank unites its loan approval authority and external risk assessment competences in a single loan service composition (LNS). The bank is interested in that the loan service composition conforms to the following two application-dependent functional requirements (behavioral claims): First, loan applications with a high amount must be investigated in detail because, according to Basel II, each loan has to be guaranteed with a risk-dependent amount of equity. Second, in order to satisfy its customers' needs, each loan application of less than 10,000 Euros and low risk should be granted.

Technically speaking, the WS composition (LNS) coordinates the functionality of two Web services (APP and ASS), and is used by a third Web service (BIZ). Three message types are necessary: request messages (*req*) that contain the requested amount of money (*amount*), approval messages (*app*) that contain the bank's decision (*result*), and risk assessment messages (*ass*) that contain the risk classification (*risk*).

In the approach of Fu et al., WS compositions are modeled by means of guarded finite state automata (GFSA). Informally speaking, finite state automata consist of states and transitions. States store information about the past. Transitions convey automata from one state to another upon external stimuli, e. g. sent or received messages. In order to deal with the content of messages, send-transitions are annotated with guards. Analogous to production rules, each guard consists of a condition part and an action part. The former specifies the transition's precondition. The latter specifies the content of the message being sent. Guards are formalized by means of XPath (Clark and DeRose 1999). Receive-transitions are not guarded because the content of received messages cannot be controlled.

Fig. 1 shows how the example can be modeled by means of GFSA. We use the standard notation for automata. States are modeled as circles, transitions as directed edges. Final states are modeled as two concentric circles, initial states as circles with edges that point to them from "nowhere". Each transition has two annotations. The first annotation indicates which message is currently being sent (!) to or received (?) from which automaton. We use indexes to distinguish several message instances of the same type (e. g. req_1, req_3). The second annotation (in squared brackets) specifies the guard. We use apostrophes to characterize that messages are forwarded ($req_3' = req_1$) or new values are assigned to variables (e. g. $app_1'.risk$ = "low"). For each WS composition and Web service, there is an automaton. Let us, for instance, consider the risk assessor's automaton (ASS). In its initial state, the automaton is waiting for a request message (req_3) forwarded by the loan service composition (?req_3←LNS). After that, the assessors' risk classification is returned to the loan service composition (LNS) via a risk assessment message (ass_1) (!ass_1→LNS). As the result can be "high" or "low", the corresponding transition indicates both possibilities. As ass_1 can always be sent, the guard condition is true.

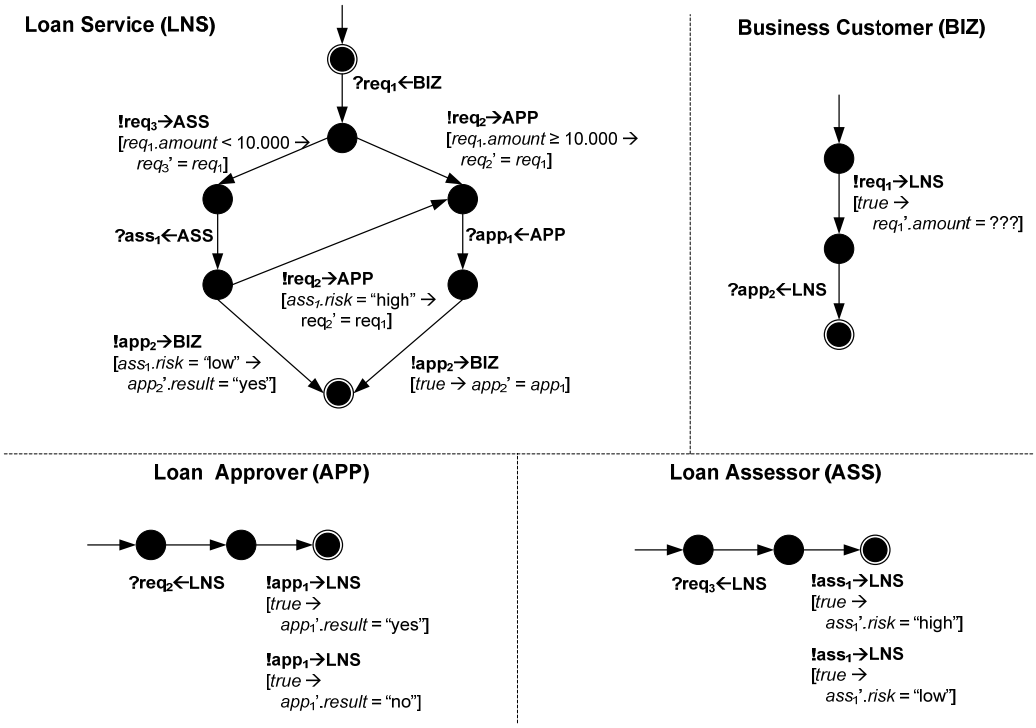

Fig. 1 The example scenario modeled with guarded finite state automata (GFSA)

Behavioral claims are formalized by means of Linear Time Temporal Logic (LTL), which counts among temporal logics (see section 2). LTL extends propositional logic by temporal operators. These indicate how propositional expressions (e. g. specific variable assignments) cohere in time (Holzmann 2003, p. 135). Two exemplary temporal operators – which will be useful below – are *globally (G)* and *eventually (F)*. The former requires that the propositional expression to which it refers remains true throughout the run of the automata (i. e. the execution of the WS composition). The latter requires that the propositional expression to which it refers becomes true at least once during the run of the automata. As each LTL formula can be transformed into a GFSA (Vardi and Wolper 1985, p. 332), it is useful to use both approaches together.

Let us, for instance, formalize the second behavioral claim from above. It requires req_1 to contain an amount of less than 10,000 Euros ($req_1.amount < 10{,}000$), ass_1 to indicate "low" risk ($ass_1.risk = \text{“low”}$), and app_2 to indicate acceptance ($app_2.result = \text{“yes”}$). Temporally speaking, the claim must hold throughout the entire execution. Thus, it must be globally (G) true. The fact that req_1 and ass_1 lead to app_2 is modeled as implication ($\rightarrow$). Although it is not known when exactly app_2 is returned, it must eventually (F) be returned. Together, these considerations result in the following LTL formula: $G(req_1.amount < 10{,}000 \wedge ass_1.risk = \text{“low”} \rightarrow F(app_2.result = \text{“yes”}))$. This formula can be translated into an automaton and serve – together with the automata from above – as input for the model checking tool SPIN, which analyzes whether the claims holds or not.

How does the approach of Fu et al. conform to the requirements framework from above? With respect to formal foundations, GFSA provide formal syntax and semantics that cover the behavioral facets of WS compositions. LTL enables to formalize both application-independent and application-independent behavioral claims. It also complies with GFSA. As for modeling support, the approach provides a verification-centered process model that is

implemented by the proprietary tool WSAT (Web Service Analysis Tool) (Fu et al. 2004a). This tool, however, does not enable to model WS compositions and specifications, neither separately nor jointly. Both must be modeled by hand. Modelers have to cope with the complexity on their own. The employed model checking tool SPIN provides constructive feedback. This feedback, however, is presented in a technical way and is only hardly suitable for conceptual modelers. Summing up, according the requirements framework, the approach could be improved in the following ways: First, the tool should enable to model behavioral claims together with WS compositions (ideally in a graphical manner). Second, this tool should be integrated with the verification tool so that the feedback of the verification process can be integrated with the representation of WS compositions.

It may be stated that the analysis leads to reasonable results. Each requirement could be assessed. It could be pointed out how the approach of Fu et al. can be improved. This corroborates our hypothesis from above – at least basically and in the sense of principle applicability.

5 Summary and Further Research

We addressed the research gap with respect to how correctness can be operationalized for WS compositions and how service-oriented modeling techniques should be shaped so that correctness can be shown by verification and WS compositions can be modeled intuitively. We propose that correctness splits into structural and behavioral correctness. The former requires the interfaces of WS compositions and component Web services to match with respect to operations and parameters. The latter requires the behavior of WS compositions to conform to specifications of application-independent and application-dependent behavioral claims. The proposed requirements framework covers the perspectives “formal foundations” and “modeling support”. The first perspective requires formal syntax and semantics for models of WS compositions and a compatible formalism for behavioral claims. The second perspective requires a process model as well as modeling tools that reduce modeling complexity, visualize the behavior of WS compositions, integrate models of compositions and specifications, and integrate the feedback of verification tools. The requirements framework has been basically evaluated by analyzing an example approach.

The results will be subject to the following research:

1. The framework comprises requirements on a conceptual level. It has only been assessed for one example approach how it could be improved, i. e. refined or extended, in order to meet the requirements. This is where further research in the sense of a comprehensive survey would be useful.

2. The requirements framework focuses on formal foundations and modeling support. It does not provide an economic perspective on verification. Showing correctness leads to overhead. This is because specifications have to be created, models have to be verified, and specifications and/or models may have to be modified repeatedly. However, for many use cases it cannot be stated in advance whether the utility realized by preventing erroneous WS compositions justifies this overhead. This economic trade-off constitutes an interesting field of further research.

References

Arjansani, Ali (2004): Service-oriented modeling and architecture. How to identify, specify, and realize services for your SOA. http://www.ibm.com/developerworks/library/ws-soa-design1/, accessed 2008-12-29.

Arkin, Assaf; Askary, Sid; Fordin, Scott; Jekeli, Wolfgang; Kawaguchi, Kohsuke; Orchard, David; Pogliani, Stefano; Riemer, Karsten; Struble, Susan; Takacsi-Nagy, Pal; Trickovic, Ivana; Zimek, Sinisa (2002): Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/2002/NOTE-wsci-20020808, accessed 2008-12-29.

Andrews, Tony; Curbera, Francisco; Dholakia, Hitesh; Goland, Yaron; Klein, Johannes; Leymann, Frank; Liu, Kevin; Roller, Dieter; Smith, Doug; Thatte, Satish; Trickovic, Ivana; Weerawarana, Sanjiva (2003): Business Process Execution Language for Web Services. Version 1.1. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, accessed 2008-12-29.

Alves, Alexandre; Arkin, Assaf; Askary, Sid; Barreto, Charlton; Bloch, Ben; Curbera, Francisco; Ford, Mark; Goland, Yaron; Guízar, Alejandro; Kartha, Neelakantan; Liu Canyang; Khalaf, Rania; König, Dieter; Marin, Mike; Mehta, Vinkesh; Thatte, Satish; van der Rjin, Danny; Yendluri, Prasad; Yiu, Alex (2007): Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, accessed 2008-12-29.

Alonso, Gustavo; Casati, Fabio; Harumi, Kuno; Machiraju, Vijay (2004): Web Services. Concepts, Architectures, Applications. Springer, Berlin.

Balzert, Helmut (1998): Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Heidelberg.

Booth, David; Haas, Hugo; McCabe, Francis; Newcomer, Eric; Champion, Michael; Ferris, Chris; Orchard, David (2004): Web Services Architecture. http://www.w3.org/TR/ws-arch/, accessed 2008-12-29.

Christensen, Erik; Curbera, Francisco; Meredith, Greg; Weerawarana, Sanjiva (2001): Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, accessed 2008-12-29.

Clark, James; DeRose, Steve (1999): XML Path Language (XPath). http://www.w3.org/TR/1999/REC-xpath-19991116, accessed 2007-12-29.

Clarke, Edmund M.; Grumberg, Orna; Peled, Doron A. (2001): Model Checking. 3rd edition, MIT Press, Cambridge.

Dostal, Wolfgang; Jeckle, Mario; Melzer, Ingo; Zengler, Barbara (2005): Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis. Spektrum, Heidelberg.

Dubray, Jean-Jaques (2008): BPML. http://www.ebpml.org/bpml.htm, accessed 2008-12-29.

Erl, Thomas (2004): Service-oriented architecture. A field guide to integrating XML and Web Services. Prentice Hall, Upper Saddle River.

Fahland, Dirk; Reisig, Wolfgang (2005): ASM-based semantics for BPEL: The negative control flow. In: *Borger, Egon.; Beauquier, Danièle; Slissenko, Anatol* (Eds.): Proceesings of the 12th International Workshop on Abstract State Machines. Paris, pp. 131-151.

Farahbod, Roozbeh; Glässer, Uwe; Vajohollahi, Mona (2004): Specification and validation of the business process execution language for Web services. In: *Zimmermann, Wolf; Thalheim, Bernhard* (Eds.): Abstract State Machines. Lecture Notes in Computer Science, Springer, pp. 79-94.

Farahbod, Roozbeh; Glässer, Uwe; Vajihollahi, Mona (2005): A Formal Semantics for the Business Process Executions Language for Web Services. In: Joint Workshop on Web services and Model-Driven Enterprise Information Systems. Miami, pp. 122-133.

Ferrara, Andrea (2004): Web Services: A process algebra approach. In: Proceedings of the 2nd International Conference on Service Oriented Computing. New York, pp. 242-251.

Ferstl, Otto K.; Sinz, Elmar J. (2006): Grundlagen der Wirtschaftsinformatik. 5th edition, Oldenbourg, München.

*Floyd, Robert W. (*1967): Assigning Meaning to Programs. In: Proceedings of the 19th American Mathematical Society Symposium in Applied Mathematics. Providence, pp. 19-32.

Fu, Xiang; Bultan, Tevfik; Su, Jianwen (2004): Analysis of Interacting BPEL Web Services. In: 13th International WWW Conference. New York, pp. 621-630.

Fu, Xiang; Bultan, Tevfik; Su, Jianwen (2004a): WSAT: A Tool for Formal Analysis of Web Services. In: 16th International Conference of Computer Aided Verification, Boston, p. 510-514.

Fu, Xiang; Bultan, Tevfik; Su, Jianwen (2005): Synchronizability of Conversations among Web Services. In: IEEE Transactions on Software Engineering 31 (12), pp. 1042-1055.

Fu, Xiang; Bultan, Tevfik; Su, Jianwen (2006): Analyzing the Conversations of Web Services. In: IEEE Internet Computing 10 (1), pp. 18-25.

Herman, Ivan (2003): Introduction to the Semantic Web. http://www.w3.org/2003/Talks/0624-BrusselsSW-IH/, accessed 2008-12-29.

Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo; Ram, Sudha (2004): Design Science in Information Systems Research. In: MIS Quarterly 28 (1), pp. 75-105.

Hoare, Charles A. R. (1969): An Axiomatic Basis for Computer Programming. In: Communications of the ACM 12 (10), pp. 576-583.

Holzmann, Gerard J. (2003): The SPIN Model Checker. Primer and Reference Manual. Addison-Wesley, Boston.

Kallus, Michael (2004)*:* Web Services vor dem Durchbruch. http://www.cio.de/news/802604/index1.html, accessed 2007-12-29.

Kavantzas, Nickolas; Burdett, David; Ritzinger, Gregory; Fletcher, Tony; Lafon, Yves; Barreto, Charlton (2005): Web Services Choreography Description Language. Version 1.0. http://www.w3.org/TR/ws-cdl-10/, accessed 2008-12-29.

Kopp, Oliver; Frenkler, Carsten; Lohmann, Niels (2006): Korrektheit und Zuverlässigkeit zusammengesetzter Web Services am Beispiel der Geschäftsprozess-Modellierungssprache BPEL. ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2006-67/INPROC-2006-67.pdf, accessed 2008-12-29.

Klyne, Graham; Carroll, Jeremy J. (2004): Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210, accessed 2008-12-29.

Lee, KangChang; Jeon, JongHong; Lee, WonSeok; Jeong, Seong-Ho; Park, Sang-Won (2003): QoS for Web Services. Requirements and Possible Approaches. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, accessed 2008-12-29.

Lohmann, Niels (2007): A feature-complete Petri net semantics for WS-BPEL 2.0. In: *van Hee, Kees; Reisig, Wolfgang; Wolf, Karsten* (Eds.): Proceedings of the Workshop on Formal Approaches to Business Processes and Web services. Podlasie, pp. 31-25.

Martens, Axel (2005): Analyzing Web Service based Business Processes. In: *Cerioli, Maura* (Ed.): Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, Springer, pp. 19-33.

Martin, David; Burstein, Mark; Hobbs, Jerry; Lassila, Ora; McDermott, Drew; McIlraith, Sheila; Narayanan, Srini; Paolucci, Massimo; Parsia, Bijan; Payne, Terry; Sirin, Evren; Srinivasan, Naveen; Sycara, Katia (2004): OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/, accessed 2008-12-29.

Mitra, Nilo (2003): SOAP Version 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, accessed 2008-12-29.

Moser, Simon; Martens, Axel; Görlach, Katharina; Amme, Wolfram; Godlinski, Artur (2007): Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In: Proceedings of the IEEE International Conference on Services Computing. Salt Lake City, pp. 98-105.

Newcomer, Eric; Lomow, Greg (2005): Understanding Service-oriented Architecture with Web Services. Addison-Wesley Longman, Amerstdam.

Papazoglou, Michael P.; van den Heuvel, Willem-Jan (2006): Service-oriented design and development methodology. In: International Journal of Web Engineering and Technology 2 (4), pp. 412-442.

Peltz, Chris (2003): Web Services Orchestration and Choreography. In: Computer 36 (10), pp. 46-52.

Rozinat, Anne; van der Aalst, Wil M. P. (2008): Conformance checking of processes based on monitoring real behavior. In: Information Systems 33 (1), pp. 64-95.

Schlingloff, Holger; Martens, Axel; Schmidt, Karsten (2005): Modeling and Model Checking Web services. In: Electronic Lecture Notes in Computer Science: Issue on Logics and Communication in Multi-Agent Systems (126). Berlin, pp. 3-26.

Schneider, Klaus (2004): Verification of reactive systems. Formal Methods and Algorithms. Springer, Berlin.

Stahl, Christian (2005): A Petri Net Semantics for BPEL. http://www2.informatik.hu-berlin.de/Institut/struktur/systemanalyse/preprint/stahl188.pdf, accessed 2008-12-29.

Sommerville, Ian (2004): Software Engineering. 7th edition, Pearson Addison-Wesley, Boston.

Van Breugel, Franck; Koshkina, Maria (2006): Models and Verification of BPEL. http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf, accessed 2008-12-29.

Vardi, Moshe Y.; Wolper, Pierre (1986): An automata-theoretic Approach to automatic Program Verification. In: First Symposium on Logic in Computer Sciences. Cambridge, pp. 332-344.

Wilde, Thomas; Hess, Thomas (2007): Forschungsmethoden der Wirtschaftsinformatik. Eine empirische Untersuchung. In: WIRTSCHAFTSINFORMATIK 49 (4), pp. 280-287.

Zimmermann, Olaf; Krogdahl, Pal; Gee, Clive (2004): Elements of Service-Oriented Analysis and Design. An interdisciplinary modeling approach for SOA projects. http://www.ibm.com/developerworks/library/ws-soad1/, accessed 2008-12-09.