Semantic Service Modeling: Enabling System Interoperability

Stanislav Pokraev[1], Dick Quartel[2], Maarten W. A. Steen[1] and Manfred Reichert[2]

[1]Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
{Stanislav.Pokraev, Maarten.Steen}@telin.nl , http://www.telin.nl
[2]Center for Telematics and Information Technology (CTIT), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{D.A.C.Quartel, M.U.Reichert}@ewi.utwente.nl, http://www.utwente.nl

Keywords: service modeling, semantic interoperability

1.1 Introduction

The integration of software systems is a major challenge for companies today. Both external forces, such as *business process integration*, and internal forces, such as the move towards *service-oriented architectures*, put increasing pressure on software engineers to reuse and integrate existing system services, rather than building new systems from scratch. However, the lack of interoperability of software systems forms a major stumbling block for the integration of such services. Hence a lot of effort is currently being invested in standardization of service description languages and protocols for service interactions [2][9][12]. Unfortunately, these efforts mainly address what we call *syntactic* interoperability, with *semantic* interoperability just starting to be addressed in a number of initiatives (see next section). These initiatives propose semantically-rich service models, definition of mappings among these models, and runtime mediation based on the defined mappings.

Interoperability is the capability of different systems to use each other's services effectively. It is about sharing functionality and information between systems at different levels, e.g., between physical devices, software applications, business units within one organization, or between different organizations. Interoperability implies that systems are able to interact (i.e., exchange messages), read and understand each other's messages, and share the same expectations about the effect of the message exchange.

In this paper we analyze and define in detail what it means for software systems to be interoperable. We identify three different levels of interoperability – the *syntactic*, *semantic* and *pragmatic* level – and define the requirements for assessing interoperability at each of these levels. We propose a method for formally verifying the semantic and pragmatic interoperability of a number of systems, given a target for integration. The latter qualification becomes necessary because interoperability is not an absolute measure. Before one can assess the interoperability of a number of services, it is necessary to define the task or the goal that these services should accomplish in concert. In other words, interoperability can only be defined with respect to the desired goal of their composition.

The goal of this paper is to explain what interoperability means and how it can be achieved. To realize this goal, we first define a number of concepts and use them to explain what a service is, how systems interact and, how they use each other’s services. Next, we discuss what interoperability problems arise when systems interact and for what reasons. Then, we identify a number of requirements for interoperability. Finally, we present a method for checking if the design of a composite system meets the requirements for interoperability.

1.2 Related Work

The Web Service Modeling Ontology (WSMO) [4] has been proposed by the SDK cluster of EU FP6 projects as an alternative for OWL-S [13]. They argue that OWL-S is only a formalization of WSDL[9] and BPEL4WS[2], and that true semantic web services require a much richer ontology. In addition to the WSMO ontology also a Web Service Modeling Language (WSML) [5] and a Web Service Execution Environment (WSMX)[6] have been defined. The objective of these specifications is to allow automatic tasks (e.g., discovery, selection, composition, mediation, execution and monitoring) to be performed with respect to services in the context of Web and grid. To solve the interoperability problems, WSMO defines *Mediators* - elements that aim to overcome structural, semantic or conceptual mismatches between the different components that build up a WSMO description.

The Semantic Web Services Framework[3] is a relatively new initiative, which addresses interoperability by proposing a language and ontology for specifying the semantics of Web Services. The language consists of two parts, namely, a first order logic language for describing Web Services (SWSL-FOL) and a rule-based language with non-monotonic semantics (SWSL-Rules). SWSL-FOL is used to formally specify service characteristics whereas SWSL-Rules is used to reason about those characteristics and execute services. SWSF also defines a formal ontology for representing service characteristics called First-Order Logic Ontology for Web Services (FLOWS).

METEOR-S project[15] is concerned with the complete lifecycle of semantic and dynamic web processes. It proposes a framework that has two main components – a configuration module and execution environment. The configuration module uses semantic service annotations based on WSDL-S[1] and constraint analysis to discover services and configure the process. The execution environment takes the output of the configuration module and handles the interactions between respective services at runtime. Data and process heterogeneities are dealt with by using a proxy with mediation capabilities.

1.3 Conceptual Framework

In this section we present our conceptual framework for service modeling. We consider the interaction between systems from a *communication*, *behavioral* and *information* perspective. For each perspective, we present concepts to model system interactions. The presented concepts are based on earlier work [16][17].

The ***communication perspective*** is concerned with modeling the interacting systems and their interconnection structure. We introduce the following two concepts for that purpose: an *Entity* models the existence of some system, while abstracting from its properties; an *Interaction point* models the existence of some mechanism that enables interaction between two or more systems, while abstracting from the properties of the mechanism. In general, the interaction mechanism is identified by its location (e.g., the combination of an IP address and port number can be used to identify a TCP/UDP socket). Entities and interaction points can further be refined to describe more precisely the internal structure of a system.

The ***behavioral perspective*** is concerned with modeling the behavioral properties of a system, i.e., the *activities* that are performed by the system as well as the *relations* among these activities. For that purpose we introduce the following basic concepts: An *Action* models an activity performed by a single entity; An *Interaction* models an activity performed by two or more entities in cooperation; an *Interaction contribution* models the contribution of an entity to an interaction; a *Causality relation* models how an action or interaction contribution depends on other actions or interaction contributions.

An Action represents a unit of activity that either occurs (completes) or does not occur (complete) during the execution of a system. Furthermore, an action only represents the activity result (effect) that is established upon completion, and abstracts from the way this result is achieved.

An Interaction represents a common activity of two or more entities. An interaction can be considered as a refinement of an action, defining the contribution of each entity involved in the interaction. Therefore, an interaction inherits the properties of an action. In addition, an interaction either occurs for all entities that are involved, or does not occur for any of them. In case an interaction occurs, the same result is established for all involved entities.

An Interaction contribution represents the participation (or responsibility) of an entity that is involved in an interaction. An interaction can only occur if each involved entity can participate. An entity can participate if the causality condition of its interaction contribution is satisfied (see below). In addition, an interaction contribution may define constraints on the possible results that can be established in the interaction. This means that an interaction represents a *negotiation* among the involved entities, only defining the possible results of the interaction, while abstracting from how they are established. We distinguish the following basic types of negotiation between two entities A and B:

- *value checking*: entity A proposes a single value x as interaction result and entity B proposes a single value y. The interaction can only occur if x = y, in which case the interaction result is x.

- *value passing*: entity *A* proposes a single value *x* as interaction result and entity *B* accepts a set of values *Y*. The interaction can only occur if $x \in Y$, in which case the interaction result is *x*.

- *value generation*: entity *A* accepts a set of values *X* as interaction result and entity *B* accepts a set of values *Y*. The interaction can only occur if $X \cap Y \neq \varnothing$, in which case the interaction result is a value from the intersection of *X* and *Y* (while abstracting from the choice of the particular value).

A Causality relation defines for an action or interaction contribution, say *a*, its causality condition, which must be satisfied to enable the occurrence of *a*. Three basic causality conditions are distinguished: *enabling condition b*, which defines that *a* depends on the occurrence of *b*, i.e., *b* must have occurred before *a* can occur; *disabling condition* $\neg b$, which defines that *a* depends on the non-occurrence of *b*, i.e., *b* must not have occurred before nor simultaneously with *a* to allow the occurrence of *a*; *start condition* $\surd$, which defines that *a* is allowed to occur from the beginning of the behavior, independent of any other actions or interaction contributions.

The basic conditions can be combined to represent more complex causality conditions using the AND- and OR-operator, which define that a conjunction and disjunction of conditions must be satisfied, respectively.

The ***information perspective*** is concerned with modeling the information that is exchanged in the interaction between the system and its environment. The subject domain of a system comprises the entities and phenomena in the real world that are *identifiable* by the system. The information model is a model of this subject domain consisting of *individuals* that represents the entity and phenomena from the subject domain, *classes* that represent the types of the entities and phenomena, and the possible relations between them. In addition, an action or interaction contribution may require that the subject domain is in a certain state before and after the occurrence of that action or interaction contribution. To model the information aspect of a system we introduce the following basic concepts: an *Individual* represents a entity or phenomenon in the system's subject domain, e.g., "*John Smith*", "*TopTech Company*" or "*London*"; a *Class* represents an abstract type of entities or phenomena in the system's subject domain, e.g., "*Person*", "*Company*" or "*City*"; a *Property* represents a relationships between entities or phenomena in the system's subject domain, e.g., "*works for*", "*is a*" or "*has office in*"; a *Result constraint* models a condition on the result of an action or interaction contribution that must be satisfied after the occurrence of the action or interaction contribution; a *Causality constraint* models a condition on the results established in causal predecessors (i.e., actions or interaction contributions) that must be satisfied to enable the occurrence of an action or interaction contribution. In this paper we use Description Logics[7], more specifically OWL-DL[8] to represent our information concepts by a concrete formalism.

Putting together the three modeling perspectives yields an ***integrated service model***. A *service* is a set of related interactions between the system and its environment. The interaction contributions are adorned with result constraints expressing the respective information constraints of the system and the environment on the values that can be established in the interactions. Taken together the interactions, their causal relations and the information constraints define the service between the system and the environment.

Our definition of service does not include a sense of ownership or initiative. An interaction is performed by two or more entities in cooperation, while abstracting from which entity initiates the interaction. However, it is often useful to talk about the service that is *offered* by a system without having to specify the constraints of the environment. Likewise, it is often useful to talk about the service that is *required* by an entity without making assumptions about the constraints of the service provider. These are two complementary views on a service, which can be obtained by only specifying one entity's contributions and constraints.

1.4 Levels of Interoperability

Software systems manage a domain of *lexical items*. These items represent entities and phenomena in the *subject domain* of the systems, e.g., patients, medicines or treatments.

Software systems interact by exchanging *messages*. Messages that enter the system request or update the state of its lexical domain. Messages that leave the system request information about the system's subject domain or provide information about the lexical domain of the system.

Messages consist of *data* that represent property values of entities or phenomena in the subject domain. The data in the messages have meaning only when interpreted in terms of the subject domain model of the system.

This research focuses on the interoperability of software systems. At this level, we distinguish between three different types of interoperability:

Syntactic interoperability is concerned with ensuring that data in the exchanged messages is in compatible formats. The message sender encodes data in a message using syntactic rules, specified in some grammar. The message receiver decodes the received message using syntactic rules defined in the same or some other grammar. Syntactic interoperability problems arise when the sender's encoding rules are incompatible with the receiver's decoding rules and this leads to the construction of mismatching message parse trees.

Semantic interoperability is concerned with ensuring that the exchanged information has the same meaning for both message sender and receiver. The data in the messages have meaning only when interpreted in terms of the respective subject domain models. However, the message sender does not always know the subject domain model of the message receiver. Depending on its knowledge, the message sender makes assumptions about the subject domain model of the receiver and uses this assumed subject domain model to construct a message and communicate. Semantic interoperability problems arise when the (assumed) sender's subject domain model differs from the receiver's subject domain model.

Pragmatic interoperability is concerned with ensuring that the message sender and receiver share the same expectation about the effect of the exchanged messages. When a system receives a messages it changes its state, sends a message back to the environment, or both[19]. In most cases, messages sent to the system change or request the system state, and messages sent from the system change or request the state of the environment. That is, the messages are always sent with some intention for achieving some desired effect. In most of the cases the effect is realized not only by a single message but by a number of messages send in some order. Pragmatic interoperability problems arise when the intended effect differs from the actual effect.

1.5 Requirements for Semantic and Pragmatic Interoperability

Web Services standards address syntactic interoperability by providing XML-based standards such SOAP[14], WSDL[9] and BPEL4WS[2]. XML is a platform-independent markup language capable of describing both data and data structure. This way, different systems can parse each other's messages, check if these messages are well-formed, and validate if the messages adhere to a specific syntactic schema. In our approach we adopt XML to deal with the syntactic interoperability and only focus on semantic and pragmatic interoperability.

Software systems exchange messages that consist of property values of entities or phenomena in their shared subject domain. Semantic interoperability problems arise when the message sender and receiver have a different conceptualization or use a different representation of the same entity type, property (type-level conflicts) or property value (value-level conflicts).

Requirement 1 *A necessary condition for semantic interoperability of two systems is the existence of a translation function that maps the entity types, properties and values of the subject domain model of the first system to the respective entity types, properties and values of the subject domain model of the second system.*

The class of possible results of an interaction is defined by the conjunction of result constraints of all contributing systems.

Requirement 2 *A necessary condition for pragmatic interoperability of a single interaction is that at least one result which satisfies the constraints of all contributing systems can be established.*

This requirement is illustrated in Figure 1. For example, if an interaction contribution defines that the result of the interaction should be a "female patient" and the other interaction contribution defines that that the result should be a "male patient", there are no possible results of this interaction.

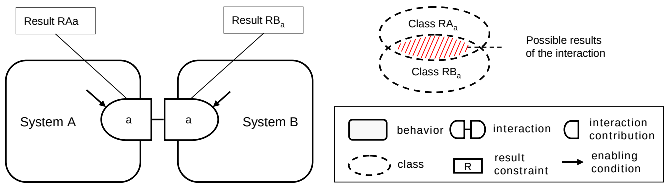

Figure 1. Pragmatic interoperability of an interaction

Requirement 3 A necessary condition for pragmatic interoperability of a service is that Requirement 2 is met for all of its interactions and they can occur in a causal order, allowed by all participating systems.

This requirement is illustrated by Figure 2. In the example, the service requestor requires that the result Ra is established first, then the result Rb, and then the result Rc. The service provider requires that the result Ra is established first, then either the result Rb (followed by Rc), or Rc (followed by Rb). If all results are possible (i.e., the interactions meet **Requirement 2**), the systems are interoperable, because the order (Ra, Rb, Rc) meets the requirements of both the service requestor and provider.

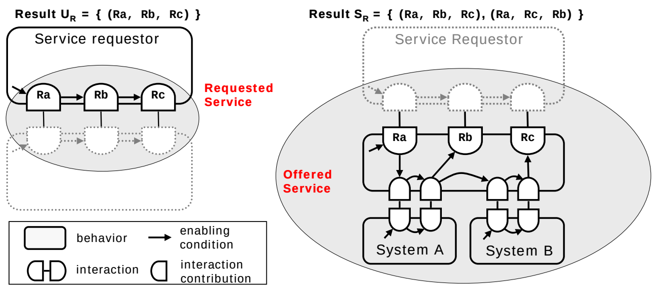

Figure 2. Pragmatic interoperability of a set of interactions

1.6 Addressing the Interoperability Requirements

To address ***Requirement 1*** we need a method to establish a mapping between *individuals*, *classes* and *properties* from the subject domains of the systems being integrated. Tools exist that use sophisticated heuristic algorithms to discover possible mappings and provide mechanisms for specifying these mappings. Besides mapping there are two other relevant approaches: *alignment* and *merging* of the subject domain models. Alignment is the process of making the subject domain models consistent and coherent with one another while keeping them separate. Merging is the process of creating a single subject domain model that includes the information from all source subject domain models.

To address ***Requirement 2***, we use Description Logic[7] as a representation system for individuals, classes, properties, result constraints and causality constraints. This way, we can describe the subject domains of the system, define classes that represent the conditions and results of actions and interaction contributions and reason if these classes can have instances or not.

The basic reasoning task in OWL-DL is *subsumption check* – a task of checking if a class *D* is more general than a class *C*. In other words, subsumption is checking if the criteria for being individual of type *C* imply the criteria for being individual of type *D*. The concept *D* is called *subsumer* and the concept *C* is called *subsumee*. Checking satisfiablity is a special case of subsumption reasoning. In this case the subsumer is the empty class (⊥). If a class *C* is subsumed by the empty class we say that the concept *C* is *not satisfiable*. This means that no individual can be of type *C*.

To address ***Requirement 2*** we define a class as an intersection of the classes that define the admissible results of an interaction for all participating interaction contributions, and check if the concept that represents the class is satisfiable.

To address ***Requirement 3*** we translate a model of a composite service described in our language to a Coloured Petri Net (CP Net) ([10][11]. The mapping is based on previous work[18].

A classical Petri net consists of a set of *places* (represented by circles), a set of *transitions* (represented by black bars), *directed arcs* connecting places to transitions or transitions to places, and *markings* assigning one or more tokens (represented by black dots) to some places. CP Nets extend the classical Petri nets by providing a mechanism for associating a *value* of a certain type to each token. In addition, a transition can be enabled only if its input tokens satisfy certain conditions (*guards*) and produce output tokens that represent new values (*bindings*). In this way, a transition can be seen as a function that maps input values to output values in a certain context.

Once we translate a service model to a corresponding CP Net, we can construct the occurrence graph of that net and reason about the dynamic properties of the model. We address ***Requirement 3*** by checking for the existence of a marking in which the results defined by the participating systems can be established. Next, we check if the order of the results establishment meets the causality constraints of the participating systems.

1.7 Conclusions

In this paper we have outlined a method for formally verifying the interoperability of a number of system services to achieve a particular task. To this end we first analyzed and defined what it means for software systems to be interoperable. We identified three different levels of interoperability – the syntactic, semantic and pragmatic level – and defined the requirements for assessing interoperability at each of these levels. Since we feel that the syntactic interoperability is sufficiently addressed by existing standards and initiatives, our method focuses on the semantic and pragmatic interoperability requirements.

Our method involves the use of a new modeling technique for services, which is more abstract and more complete than most existing service description techniques (WSDL, BPEL4WS), including current proposals for the specification of Semantic Web Services (OWL-S, SWSO, WSMO). Our approach combines the *precise, but abstract definition of the behavior* of services and their compositions with a *formal definition of the information* being exchanged between services. Once we have specified services in this formalism, we are able to apply a combination of a formal logic reasoner and a formal behavior analysis tool to verify the semantic and the pragmatic interoperability of a given set of services.

1.8 Acknowledgements

The presented work has been done in the Freeband Communication project A-Muse (http://a-muse.freeband.nl). Freeband Communication (http://www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025.

The presented work is a result of a collaboration between the Telematica Instituut and the University of Twente, the Netherlands, which is supported by the Commission of the European Communities under the sixth framework (INTEROP Network of Excellence, Contract N° 508011, http://www.interop-noe.org/.

We would like to thank Henk Jonkers and Patrick Strating from the Telematica Instituut and Luis Ferreira Pires from the University of Twente for their valuable comments on this work.

1.9 References

1. Akkiraju R, Farrell J, Miller J, Nagarajan M, Schmidt M-T, Sheth A, Verma K (2005) Web Service Semantics - WSDL-S. W3C Member Submission 7 November 2005, version 1.0
2. Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Trickovic I, Weerawarana S (2003) Business Process Execution Language for Web Services, Version 1.1, 5 May 2003, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
3. Battle S, Bernstein A, Boley H, Grosof B, Gruninger M, Hull R, Kifer M, Martin D, McIlraith S, McGuinness D, Su J, Tabet S (2005) Semantic Web Services Framework (SWSF) Overview, W3C Member Submission 9 September 2005, http://www.w3.org/Submission/SWSF/
4. Bruijn Jd, Bussler C, Domingue J, Fensel D, Hepp M, Keller U, Kifer M, König-Ries B, Kopecky J, Lara R, Lausen H, Oren E, Polleres A, Roman D, Scicluna J, Stollberg M (2005) Web Service Modeling Ontology (WSMO), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSMO/
5. Bruijn Jd, Fensel D, Keller U, Kifer M, Lausen H, Krummenacher R, Polleres A, Predoiu L (2005) Web Service Modeling Language (WSML), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSML/
6. Bussler C, Cimpian E, Fensel D, Gomez JM, Haller A, Haselwanter T, Kerrigan M, Mocan A, Moran M, Oren E, Sapkota B, Toma I, Viskova J, Vitvar T, Zaremba M (2005) Web Service Execution Environment (WSMX), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSMX/

7. Calvanese D, McGuinness D, Nardi D, Schneider PP (2003) The Description Logic Handbook: Theory, Implementation and ApplicationsCambridge University Press, 2003. ISBN 0521781760. http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=0521781760
8. Dean M, Schreiber G, Bechhofer S, Harmelen Fv, Hendler J, Horrocks I, McGuinness DL, Schneider PP, Stein LA (2004) OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-ref/
9. Erik C, Curbera F, Meredith G, Weerawarana S (2001) Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl
10. Jensen K (1992) Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 1992. ISBN: 3-540-60943-1.
11. Jensen K (1994) Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods. Monographs in Theoretical Computer Science, Springer-Verlag, 1994. ISBN: 3 -540-58276-2
12. Kavantzas N, Burdett D, Ritzinger G, Fletcher T, Lafon Y (2004) Web Services Choreography Description Language Version 1.0, W3C Working Draft 17 December 2004, http://www.w3.org/TR/ws-cdl-10/
13. Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, Sirin E, Srinivasan N, Sycara K (2004) OWL-S: Semantic Markup for Web Services W3C Member Submission 22 November 2004, http://www.w3.org/Submission/OWL-S/
14. Martin G, Hadley M, Mendelsohn N, Moreau, J-J, Nielsen HF (2003) SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part1/
15. METEOR-S: Semantic Web Services and Processes, http://lsdis.cs.uga.edu/projects/meteor-s/
16. Quartel DAC, Dijkman RM, Sinderen MJv (2004) Methodological support for service-oriented design with ISDL. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York City, NY, USA, 2004.
17. Quartel DAC, Ferreira LP, Sinderen MJv (2002) On Architectural Support for Behaviour Refinement in Distributed Systems Design. In: Journal of integrated design and process science online, 06(01) issn. 1092-0617.
18. Sinderen MJv, Ferreira LP, Vissers CA, Katoen JP (1995) A design model for open distributed processing systems. Computer Networks and ISDN Systems, Vol. 27, 1995, pp. 1263-1285. ISSN 0169-7552.
19. Wieringa RJ (2003) Design Methods for Reactive Systems: Yourdon, Statemate, and the UML. Morgan Kaufmann, 2003. http://www.mkp.com/dmrs