Requirements and Method for Assessment of Service Interoperability

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

[1] Telematica Instituut, The Netherlands, P.O. Box 589
7500 AN Enschede, The Netherlands
[2] Centre for Telematics and Information Technology,
University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands
Stanislav.Pokraev@telin.nl, D.A.C.Quartel@ewi.utwente.nl,
Maarten.Steen@telin.nl and M.U.Reichert@ewi.utwente.nl

Abstract. Service interoperability is a major obstacle in realizing the SOA vision. Interoperability is the capability of multiple, autonomous and heterogeneous systems to use each other's services effectively. It is about the meaningful sharing of functionality and information that leads to the achievement of a common goal. In this paper we systematically explain what interoperability means and analyze possible interoperability problems. Further, we define requirements for service interoperability and present a method to assess whether a composite system meets the identified requirements.

Keywords: service modeling, service interoperability, formal verification

1 Introduction

The integration of software systems is a major challenge for companies today. Both organizational forces, such as *business process integration (BPI)*, and technology drivers, such as the move towards *service-oriented architectures (SOA)*, 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 forms a major stumbling block in the integration process. To address this issue a lot of efforts are currently being invested in standardizing service description languages and protocols for service interactions such as WSDL, BPEL, WS-CDL. Unfortunately, these efforts mainly address what we call *syntactic interoperability*, whereas *semantic interoperability* is just starting to be addressed by initiatives such as the SWSI[1] and the WSMO[2] working groups.

In this paper we analyze what it means for software systems to be *interoperable*. Based on the results of this analysis we identify possible interoperability problems and define requirements for appropriate solutions. Next, we propose a conceptual framework for service modeling as well as a method for formally verifying the inter-

[1] http://www.swsi.org/
[2] http://www.wsmo.org/

operability of an integrated system, starting with an integration goal. The latter qualification becomes necessary because a composite system has properties that emerge due to the interaction of its components. Assessing interoperability of such a system means that one can check if a desired goal (i.e., a number of emerging properties) can be achieved by the elements of that system in concert.

The paper is organized as follows: Section 2 presents our conceptual framework for service modeling. Section 3 explains what interoperability means, analyze possible interoperability problems, and define requirements for service interoperability. Section 4 presents our method for formal verification whether a composite system meets the identified interoperability requirements. Section 5 gives an overview of the state-of-the art and the related work. Finally, Section 6 presents our conclusions and discusses some future research directions.

2 A Conceptual Framework for Service Modeling

This section presents our conceptual framework for service modeling. The framework defines concepts and a notation to model interactions between systems from a *communication*, *behavioral* and *information* perspective. The presented concepts are generic in that they can be applied in different application domains and at successive abstraction levels. This helps limiting the number of required concepts. The core concept in our framework is the *interaction* concept. It supports a *constraint-oriented style* of service specification, which facilitates the addressing of interoperability requirements by modeling the participation of interacting entities as separate constraints and by reasoning about satisfiability of the logical conjunction of these constraints. The conceptual framework is based on earlier work [12][13].

The ***communication perspective*** is concerned with modeling the interacting systems and their interconnection structure. For that purpose we introduce two basic concepts, namely *Entity* and *Interaction point*.

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).

We adopt Webster's definition of a system, which defines a system as "*a regularly interacting or interdependent group of items, components or parts, forming a unified whole*". This definition distinguishes between two system perspectives: an *internal* perspective, i.e., the "*regularly interacting or interdependent group of items, components or parts*", and an *external* perspective, i.e., the "*unified whole*". Figure 1 illustrates both perspectives.

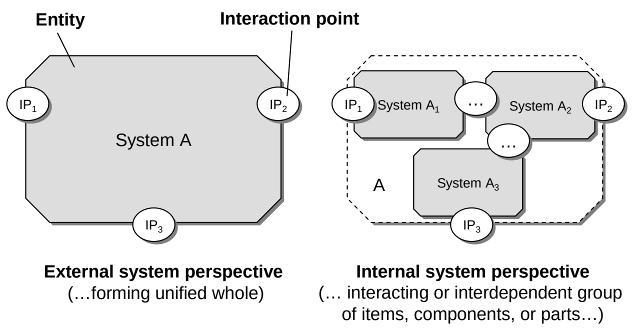
Entity
Interaction point
IP1
IP2
IP3
System A
System A1
System A2
System A3
A
…
External system perspective
(…forming unified whole)
Internal system perspective
(… interacting or interdependent group of items, components, or parts…)

Figure 1. Communication perspective

From an external perspective a system is modeled as a single entity (e.g., *System A*) having one or more interaction points (e.g., IP_1, IP_2 and IP_3). From an internal perspective a system is modeled as a structure of interconnected system parts (e.g., *Systems* A_1, *System* A_2 and *System* A_3).

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 them. For that purpose we introduce four basic concepts, namely, *Action, Interaction, Interaction contribution* and one relation, namely, *Causality relation.*

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 potential results of the interaction, while abstracting from how they are established. We distinguish three 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 $X \cap Y \neq \emptyset$, 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).

For an action or interaction contribution, say *a*, a *Causality relation* defines the condition that must be satisfied to enable the occurrence of *a*. Three basic 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 for the occurrence of *a*;
- *Start condition* √, which defines that *a* is allowed to occur from the beginning of the behavior, independent of any other actions or interaction contributions.

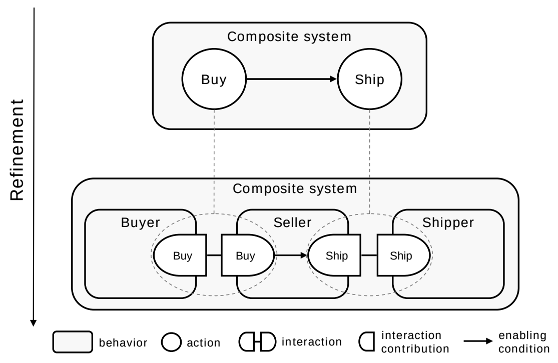

Figure 2. Refinement of an action

The behavioral concepts are illustrated in Figure 2. At the higher abstraction level action Buy is followed by action Ship. At the lower level the actions are refined by assigning actors (e.g., Buyer, Seller and Shipper) that contribute to the result of these actions. Constraints on the results of interactions that systems may define are discussed later after having introduced the information perspective.

Basic conditions can be combined to represent more complex causality conditions. For this we provide the *AND* and the *OR* operators, which define that a conjunction and disjunction of conditions must be satisfied, respectively.

The ***information perspective*** is concerned with modeling the *subject domain* of a system. First, we explain what subject domain is and then we introduce five basic modeling concepts.

Software systems manage a domain of *lexical* items. These items represent entities and phenomena in the real world that are *identifiable* by the system (e.g., people, companies or locations). In this context we denote the part of the world that is identifiable by the systems as *subject domain* of the system.

Software systems interact with their environment 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 from the subject domain. The data in the messages have meaning only when interpreted in terms of the subject domain model of the system.

To model the information perspective we provide five basic concepts, namely *Individual*, *Class*, *Property*, *Result constraint* and *Causality constraint*.

An *Individual* represents an entity or phenomenon in the subject domain of the system, e.g., the person "*John*", the hospital "*Saint Joseph*" or the city "*London*".

A *Class* represents an abstract type of entities or phenomena in the subject domain of the system, e.g., "*Patient*", "*Hospital*" or "*City*".

A *Property* represent possible relations that can exist between entities or phenomena in the system's subject domain, e.g., "*admitted to*", "*is a*" or "*is located 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.

Figure 3 shows how information concepts are related to interactions.

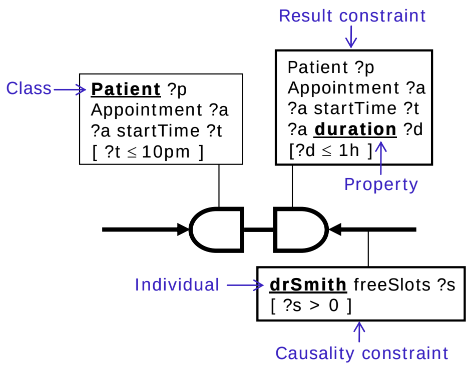

Figure 3. Relating information concepts to an interaction

In the example a system requests an appointment for a patient starting not later than 10pm. The hospital system accepts any appointments with duration less or equal than 1 hour. In addition, the interaction can only happen if Dr. Smith (the healthcare professional responsible for this case) has free slots in his calendar. Indeed, this is a causality constraint if the individual drSmith has been established as a result of a preceding (inter)action.

We use *Description Logics (DL)*[6], more specifically *OWL-DL*[7] to represent our information concepts by a concrete formalism. DL ontologies consist of *concepts*, *roles* and *individuals*. *Individuals* represent entities of phenomena from the real world, *concepts* represent abstract classes of entities or phenomena, and *roles* represent relations between entities or phenomena.

A concept can be *atomic*, i.e., denoted by its name (e.g., *Patient*, *Room* or *Hospital*) or defined as an expression that contains other concepts, roles and composition operators such as *union* or *intersection*.

Besides concepts, individuals and relations, DL ontologies consist of a set of *axioms* specifying the properties of the concepts, roles and individuals. Examples of such axioms are concept inclusion ($\mathsf{C(x)} \wedge \mathsf{C} \subseteq \mathsf{D} \rightarrow \mathsf{D(x)}$), role inclusion ($\mathsf{R(x, y)} \wedge \mathsf{R} \subseteq \mathsf{S} \rightarrow \mathsf{S(x, y)}$), transitive role ($\mathsf{R(x, y)} \wedge \mathsf{R(y, z)} \rightarrow \mathsf{R(x, z)}$), etc. For the formal semantics of OWL-DL we refer to [7].

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. An example is given in Figure 4a. It shows two interacting behaviors, representing the behaviors the one of a system and another one of its environment. These entities can engage in three interactions *a*, *b* and *c*, which are related by causality relations. The interaction contributions can be adorned with result constraints and the causality relations with causality constraints respectively. 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 direction. It is an interaction that models a common activity of two or more entities in which some results (values) can be established, but abstracts from who takes the initiative or the direction in which values flow. However, often it is useful to talk about the service that is *offered* by a system without having to specify the constraints of the environment. Likewise, it is also often useful to talk about the service that is *requested* 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 (cf. Figure 4b and Figure 4c).

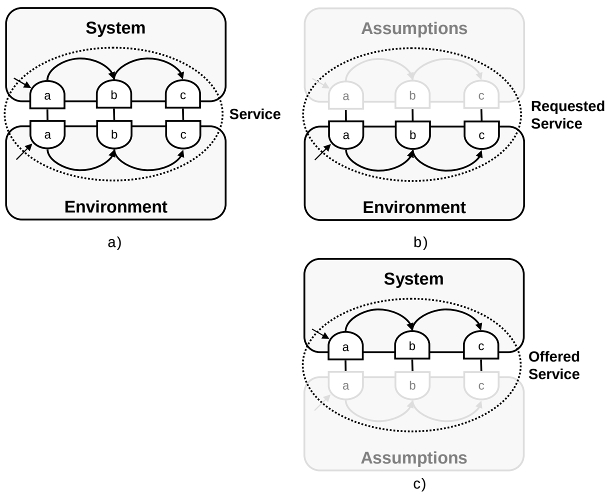

Figure 4. Service model

3 Requirements for Interoperability

In our approach we distinguish three different levels of interoperability, namely *syntactic*, *semantic* and *pragmatic*.

Syntactic interoperability is concerned with ensuring that data from the exchanged messages are 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, which leads to (construction of) mismatching message parse trees.

Web Services standards address syntactic interoperability by providing XML-based standards such as SOAP, WSDL and BPEL. 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 syntactic interoperability and only focus on semantic and pragmatic interoperability.

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 to communicate it. Semantic interoperability problems arise when the message sender and receiver have a different *conceptualization* or use a different *representation* of the entity types, properties and values from their subject domains. Examples of such differences are *naming conflicts* (same representation is used to designate different entities, entity types or properties, or different representations are used to designate the same entity, entity type or property), *generalization conflicts* (the meaning of an entity type or a property is more general than the meaning of the corresponding entity type or property), *aggregation conflicts* (an entity type aggregates two or more corresponding entity types), *overlapping conflicts* (an entity type or a property partially overlaps a corresponding entity type or a property), *isomorphism conflicts* (the same entity type or property is defined differently in different subject domain models), *identification conflicts* (the same entity is identified by different properties), *entity-property conflicts* (an entity type in modeled as a property), etc.

To address the identified semantic conflicts we define the following requirement:

Requirement 1: A necessary condition for the 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.

Pragmatic interoperability is concerned with ensuring that 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[18]. 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.

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

As said earlier, a service is a set of related interactions between the system and its environment.

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.

The requirements are discussed in more details in the next section**.**

4 Formal Verification of Service Designs

In this section we present a formal method for checking if a service design meets the requirements identified in the previous section.

To address *Requirement 1* we need a method to establish mappings between values, concepts and relations from subject domains of the systems being integrated. This method requires understanding of the meaning of values, concepts and relations from the respective subject domains and cannot be fully automated. However, 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 the semantic conflicts identified in the previous section we need a formal language capable of expressing mappings. In the following we show how some of the identified problems can be addressed using OWL-DL axioms. In the explanation below we use the prefixes `a:` and `b:` to identify a concept or a relation in the subject domain model of *System A* and *System B* respectively.

Naming conflicts can be addressed using axioms that assert sameness (e.g., `a:Medicine` $\equiv$ `b:Drug`) or difference (`a:Employee` $\neq$ `b:Employee`). Aggregation conflicts can be addressed using axioms that define a new concept as aggregation of the corresponding concepts (e.g., `a:Address` $\equiv$ `List (b:StreetNo, b:Street, b:City)`. Generalization conflicts can be addressed using axioms that assert the generalization (or specialization) relation between the respective concepts (`a:Human` $\subseteq$ `b:Patient`). Overlapping conflicts can be addressed using axioms that assert that corresponding concepts are not disjoint (e.g., $\neg$(a:Man $\cap$ b:Adult) $\subseteq \perp$).

Unfortunately, not all types of mappings can be expressed using OWL. For example, OWL does not allow for property chaining (e.g., a:hasUncle $\equiv$ b:hasBrother • b:hasFather) and qualified cardinality restrictions a:SafeBuilding = b:Building $\cap$ $\geq$2b:hasStairs.b:FireEscapeStairs which makes it difficult (in some cases impossible) to deal with isomorphic and cardinality conflicts. However, some of these issues are being dealt with in the upcoming version of OWL 1.1.

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.

As said earlier, we use OWL-DL as a representation system for individuals, classes and properties as well as to define result 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 concept D is more general than a concept 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. If C subsumes D and D subsumes C, then we can conclude that class C and D are equivalent.

Checking concept satisfiability is a special case of subsumption reasoning. In this case the subsumer is the empty concept ($\perp$). If a concept C is subsumed by the empty concept we say that the concept *C* is not satisfiable. This means that no individual can be of type C.

Requirement 2 is illustrated in Figure 5. In this example, any appointment not earlier than 10pm with duration no longer that 1 hour is a possible result of the interaction *a*.

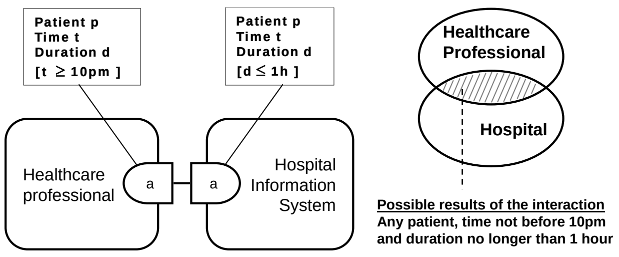
Patient p
Time t
Duration d
[t ≥ 10pm]
Patient p
Time t
Duration d
[d ≤ 1h]
Healthcare Professional
Hospital
Healthcare professional
a
a
Hospital Information System
Possible results of the interaction
Any patient, time not before 10pm and duration no longer than 1 hour

Figure 5. Example of Requirement 2

To check if a composite system meets *Requirement 3* we translate a model of a composite service described in our language to a Coloured Petri Net (CPN)[8][9]. This way we can construct the corresponding occurrence graph and reason about the dynamic properties of the model. The presented mapping is partially based on previous work [16].

A classical Petri Net (PN) 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. CPNs extend the classical PNs 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.

An action in our language maps to a transition in terms of PNs. A transition can be executed when all incoming places contain at least one token. On execution it consumes a token from all incoming places and produces a token in all outgoing places. Similar to actions, enabled transitions may execute in parallel. Nets that correspond to some elementary causality relations from our language are depicted in Figure 6:

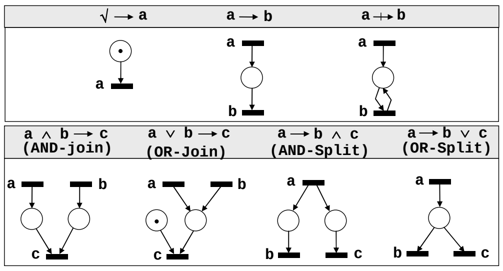

Figure 6. Mapping to Petri Nets

As said earlier, the occurrence or the result of an action (or interaction) may depend on the result of one or more causal predecessors (actions or interactions). Such dependences can be easily mapped onto guards and bindings in terms of CPNs. Figure 7 shows an example of the respective mappings.

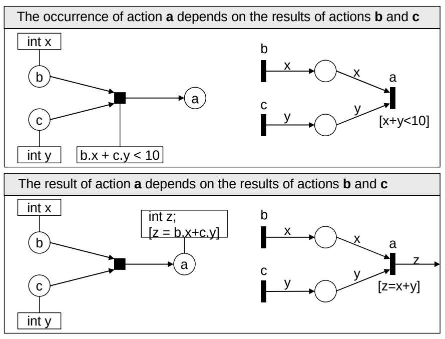
The occurrence of action a depends on the results of actions b and c
int x
int y
b.x + c.y < 10
[x+y<10]
The result of action a depends on the results of actions b and c
int z;
[z = b.x+c.y]
[z=x+y]

Figure 7. Mapping to Coloured Petri Nets

The presented mappings allow models expressed in our language to be translated into CPN and analyzed using existing tools.

To check if the composition from our example meets *Requirement 3*, we translate the model to the corresponding CPN using the presented mapping and construct the occurrence graph of that net. We use the constructed graph to check 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. The requirement is illustrated in Figure 8 and explained in an example below.

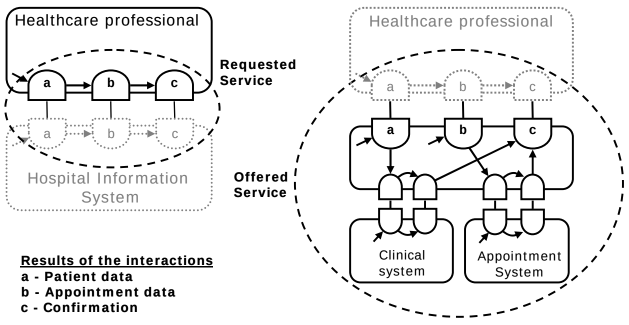

Figure 8. Example of Requirement 3

Consider a healthcare professional who wants to refer a patient to a specialist. His system allows him to send the patient's clinical data, followed by the appointment data and finally to receive a confirmation from the hospital information system. I.e., the allowed interaction order of the healthcare professional is *a, b*, *c*. The hospital information system can either first receive the patient's clinical data or the appointment data. Once it has both it registers the data in the clinical and appointment systems and sends back a confirmation to the healthcare professional. I.e., the allowed interaction order of the hospital information system is *a*, *b*, *c*, or *b*, *a, c*. In the example, the systems are interoperable because the order *a*, *b, c* meets the constraints of both the healthcare professional and the hospital information system.

To validate our conceptual framework, we implemented a prototype that checks if a composite system meets the identified requirements. Our prototype uses Racer[14] Renew[10] and CPNTools[15].

5 State-Of-the Art and Related work

OWL-S[11] is an OWL ontology for Web Services, aiming at making them computer-interpretable, to enable automatic service discovery and invocation, i.e., breaking down interoperability barriers through precise service semantics. For that reason OWL-S defines a class *Service*, where all service properties are very general. The idea is to provide a conceptual basis for building service taxonomies, but it is expected that taxonomies will be created according to functional and domain-specific needs. A service has a *ServiceProfile*. This is a high level description of the service and its provider. A *ServiceProfile* describes the functional and non-functional service properties in a human readable way. The service is formally described by a

ServiceModel. It provides means for describing the data and control flow in case of a composite service. Finally, a service has a *ServiceGrounding*, which is a specification of service access information such as communication protocols, and transport mechanisms.

IBM together with LSDIS Lab at University Of Georgia has proposed lightweight approach for adding semantics to Web Service descriptions, WSDL-S[1]. It is based in the work done in METEOR-S[17]. WSDL-S provides a mechanism to annotate WSDL service descriptions by providing extension elements such as input, output, precondition and effect. The intention is to build upon other Semantic Web Services related efforts. WSDL-S relies on both the WSDL and XML Schema extension mechanisms to reference external semantic models, without being constrained to a particular semantic representation language.

The Web Service Modeling Ontology (WSMO) [3] has been proposed as an alternative for OWL-S. The creators of WSMO argue that OWL-S is only a formalization of WSDL and BPEL4WS, and that true service semantics require a much richer ontology. In addition to the WSMO ontology also a Web Service Modeling Language (WSML) [4] and a Web Service Execution Environment (WSMX) [5] have been defined. The objective of these specifications is to allow automatic service discovery, composition, execution and interoperation in the context of Web and Grid.

The Semantic Web Services Framework [2] 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).

6 Conclusions

The main contributions of this work are the definition of a conceptual framework for service modeling, the identification of requirements for semantic and pragmatic interoperability and a method for assessing whether a composite system meets the identified requirements. We did this by first analyzing and defining 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 syntactic interoperability is sufficiently addressed by existing standards and initiatives, we focused on the semantic and pragmatic interoperability requirements.

What makes our work different from the related work in the area is that our method is based on a new service modeling framework which provides generic concepts that can be applied in different application domains and at successive abstraction levels. The key concept in our framework (the concept *Interaction)* supports a *constraint-*

oriented style of service specification. This style allows service requestors and providers to *explicitly* specify their assumptions about the environment of their systems. This in turn enables formal verification of the interoperability of the composite system by checking constraint satisfiability.

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.

There are a number of issues that we still need to address to make our method more practical.

First, we cannot assume that existing services are specified using our modeling notation. Therefore, we are working on providing mappings from existing service description languages and tools for the (semi)-automatic transformations of service models from descriptions in WSDL and BPEL.

Second, we plan to investigate ways of presenting the verification results back into the original models. Currently, the outcome of applying our method is a yes/no-answer. However, it is not very satisfactory to find out that a particular composition of services is not interoperable. In that case more feedback is required as to the cause of the interoperability problem.

Finally, we would like to investigate ways to (semi)-automatically derive mediators capable of solving detected semantic and pragmatic interoperability problems. Such mediators should implement mappings between the information and behavioral models to overcome semantic and pragmatic interoperability problems.

Acknowledgments. 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.We would like to thank Henk Jonkers, Patrick Strating and Rogier Brussee from the Telematica Instituut, the Netherlands for their valuable comments on this work.

References

1. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., Verma, K. Web Service Semantics - WSDL-S. W3C Member Submission 7 November 2005, Version 1.0, http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/
2. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S. Semantic Web Services Framework (SWSF) Overview, W3C Member Submission 9 September 2005, http://www.w3.org/Submission/SWSF/
3. Bruijn, J. de, 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. Web Service Modeling Ontology (WSMO), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSMO/

4. Bruijn, J. de, Fensel, D., Keller, U., Kifer, M., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L. Web Service Modeling Language (WSML), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSML/
5. Bussler, C., Cimpian, E., Fensel, D., Gomez, J. M., Haller, A., Haselwanter, T., Kerrigan, M., Mocan, A., Moran, M., Oren, E., Sapkota, B., Toma, I., Viskova, J., Vitvar, T., Zaremba, M. Web Service Execution Environment (WSMX), W3C Member Submission 3 June 2005, http://www.w3.org/Submission/WSMX/
6. Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003. http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=0521781760
7. Dean, M (eds.), Schreiber, G.(eds.), Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A. OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-ref/
8. Jensen, K. 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.
9. Jensen, K. 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
10. Kummer, O., Wienberg, F., Duvigneau, M., Köhler, M., Moldt, D., Rölke, H. Renew - The Reference Net Workshop. In Veerbeek, E. (editor), Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003)., pages 99-102.
11. 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. OWL-S: Semantic Markup for Web Services W3C Member Submission 22 November 2004, http://www.w3.org/Submission/OWL-S/
12. Quartel, D.A.C., Dijkman R.M., Sinderen van M. J. 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.
13. Quartel, D.A.C., Ferreira Pires, L., Sinderen, van M. J. On Architectural Support for Behaviour Refinement in Distributed Systems Design. In: Journal of integrated design and process science online, 06(01) ISNN 1092-0617.
14. Racer Systems, Racer Reasoner, http://www.racer-systems.com/, 2005
15. Ratzer, A. V., Wells, L., Lassen, H. M., Laursen, M., Qvortrup, J. F., Stissing, M. S., Westergaard, M., Christensen, S., Jensen, K. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Net, In: Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003), Eindhoven, The Netherlands, June 23-27, 2003, pages 450-462. Volume 2679 of Lecture Notes in Computer Science / Wil M. P. van der Aalst and Eike Best (Eds.) Springer-Verlag, June 2003.
16. Sinderen, M. J. van, Ferreira Pires, L., Vissers, C. A., Katoen, J.P. A design model for open distributed processing systems. Computer Networks and ISDN Systems, Vol. 27, 1995, pp. 1263-1285. ISSN 0169-7552.
17. Verma, K., Gomadam, K., Sheth, A., Miller, J., Wu, Z. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes", Technical Report . Date: 6-24-05.
18. Wieringa, R. J. Design Methods for Reactive Systems: Yourdon, Statemate, and the UML. Morgan Kaufmann, 2003. http://www.mkp.com/dmrs