Service Substitution Revisited

Dionysis Athanasopoulos, Apostolos Zarras, Valérie Issarny

Service Substitution Revisited

Dionysis Athanasopoulos
Dept. of Computer Science
University of Ioannina, Greece
dathanas@cs.uoi.gr

Apostolos V. Zarras
Dept. of Computer Science
University of Ioannina, Greece
zarras@cs.uoi.gr

Valerie Issarny
INRIA-Rocquencourt, France
Domaine de Voluceau, France
Valerie.Issarny@inria.fr

ABSTRACT

The current state of the art concerning the problem of service substitution raises the following issue: the complexity of the substitution process scales up with the number of available services that may serve as candidate substitutes for a target service. To deal with this issue, we propose a framework that is based on two substitution relations and corresponding theorems. The proposed relations and theorems allow organizing available services into groups. Then, the complexity of retrieving candidate substitute services for the target service and generating corresponding adapters scales up with the number of available groups, instead of scaling up with the number of available services.

1. INTRODUCTION

The service-oriented paradigm fosters the development of software consisting of basic architectural elements (a.k.a services) that are by themselves autonomous systems, which have been independently developed [2]. Amongst the most critical issues that should be handled towards the success of this paradigm is the independent evolution of services along with their variation in quality (e.g. performance, availability, reliability) [10, 15]. Services evolve independently; a service may be deployed, or un-deployed at anytime; its implementation, along with its interface may change without prior notification. Moreover, multiple services may be available with different quality features such as response time, MTBF, MTTR, etc. The straightforward approach for dealing with a service that is no longer available, or a service that no longer satisfies the quality requirements of the client software that uses it, is to try to substitute it with another one that implements the same interface. Substituting the target service with a another service that offers the same interface implies that changes to the client software shall be minimal; the client software should be modified to use the endpoint address of the substitute service, instead of the endpoint address of the target service.

Taking an example, suppose that the client software is a Java application that allows manipulating information about scientific publications. The application takes advantage of a Web search engine for publications that is exposed through a programmable Web service interface. The interface provides an operation, named `advanced_scholar_search`, which accepts as input a parameter `as_q` that contains search terms, a parameter `as_authors` that contains a required author name, a parameter `as_ylo` that specifies a lower bound on the publication date and a parameter `as_yhi` that corresponds to an upper bound on the publication date (Figure 1 - upper part). The service operation returns as output a document, which comprises information about publications that match the required criteria. Note that the interface of the Web service is inspired from the front-end Web interface of the GoogleScholar[1] publications search engine. The Web service of our scenario is compliant with the W3C standard[2]; therefore, it can be accessed via invocations on the `advanced_scholar_search` operation that conform to a standard RPC mechanism, such as JAXRPC. The client code for such an invocation is given in the upper part of Figure 2. Substituting the GoogleScholar service with another one that provides the same interface requires no changes in the client code; only the value of the `url` parameter should be changed (Figure 2 - upper part, line 4).

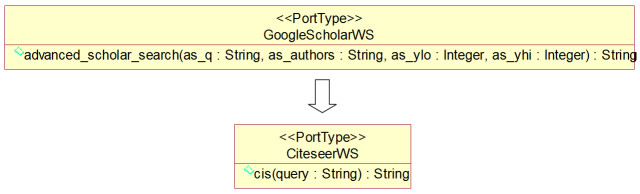


Figure 1: Substituting a service with another service that offers a different interface.

However, the aforementioned approach is not practical in the service-oriented paradigm. In general, there may be multiple similar available services. Nevertheless, these services rarely provide the same interface (e.g. check the Web Service List[3] site that maintains a large collection of services). In this case, substituting the target service, with a substitute service that implements a different interface, may trigger significant changes in the client code, increasing thus the

[1]http://scholar.google.com/advanced_scholar_search
[2]http://www.w3c.org/TR/ws-arch
[3]http://www.webservicelist.com/webservices/

WOODSTOCK '97 El Paso, Texas USA

overall maintenance effort.

Returning to our simple example, we can easily envision services that are similar with the GoogleScholar publications search engine. Typical examples of Web sites that offer similar functionality are Citeseer[4], the ACM digital library[5] and several others. Based on the Web site functionality of Citeseer, suppose that a corresponding W3C Web service offers an operation, `cis`, which accepts as input a single parameter `query` that contains search terms and returns as output a document, comprising information about publications that contain the given search terms (Figure 1 - lower part). Then, substituting GoogleScholar with Citeseer requires significant changes in the client application source code (Figure 2 - lower part, lines 3,4,7).

```
1 public class Application {
2  public static void main(String args[]){
3    GoogleScholarWSInterface ref =
4          new GoogleScholarWSSOAPBindingStub(url);
4  //.....
5  //.....
6    String publications = ref.scholar_advanced_search(
7          args[0], args[1], args[2], args[3]);
8   }
9 }
```

```
1 public class Application {
2  public static void main(String args[]){
3    CiteseerWSInterface ref =
4          new CiteseerWSSOAPBindingStub(url);
5  //.....
6  //.....
7    String publications = ref.cis(args[0]);
8   }
9 }
```

Figure 2: Maintenance effort on the side of the client application.

Research efforts that focus on service substitution can be divided in two categories, derived from two different perspectives. The first category consists of *abstraction-based* approaches that propose development methodologies and frameworks that allow *developing from scratch client applications, which use service abstractions that are mapped into alternative concrete services* [7, 16, 1]. Through a particular service abstraction, it is possible to use any of the alternative concrete services without changing the client application code. The second category comprises *adapter-based* approaches, which deal with *existing client applications that use concrete services* [14, 13, 9]. The basic concept in this case is to derive a mapping between the target service that should be substituted and a substitute service that offers similar functionality through a different interface. Based on such a mapping, an adapter is generated [5, 19]; the adapter allows accessing the functionality of the substitute service through the original target interface, without modifying the client application code. Deriving the mapping between the target service and the substitute service is typically done by examining the interface of the target service against the interfaces of a set of candidate available services for *compatibility problems* [14].

While considering *adapter-based* approaches the following issue is raised: *the complexity of the service substitution problem scales up with the number of available services that should be examined as potential candidate substitutes of the target service*. In other words, as the cardinality of available services increases, the *effort* and *time* required for the substitution of the target service also increases. This problem is a serious drawback towards a practical service substitution approach if we consider that several tasks of the service substitution process may involve human intervention (e.g. resolve interface incompatibilities). The service substitution complexity issue becomes even more important given that the difficulty, or even the impossibility of guaranteeing semantic compatibility (that the substitute service actually does what one wants [14]) usually leads us to examine the target service against all available services, towards retrieving and generating adapters for *a set of possible candidate substitute services*, instead of a single service.

In this paper, we share the objective of *adapter-based* approaches, i.e. enabling service substitution in client applications that use concrete services. However, our specific goal is to reduce the effort and time required to achieve this objective. To this end, we propose a *hybrid approach* that borrows ideas from *abstraction-based* approaches so as to handle the complexity of service substitution. The proposed approach relies on a formal foundation that comprises two substitution relations and corresponding substitution theorems, which are inline with the Liskov substitution principle (LSP) [6]. Based on the proposed relations and theorems, available services are organized into groups, characterized by *abstractions*, called *profiles*. Then, the complexity of service substitution scales up with the number of available profiles, instead of scaling up with the number of available services. Our substitution relations are similar with several variants, derived from LSP and used in the context of matching service descriptions (e.g. [12],[11],[8]). Therefore, the main thrust of our contribution is not the relations by themselves, but their combined use, along with the proposed substitution theorems, towards reducing the complexity of the service substitution problem.

The rest of this paper is structured as follows. Section 2 briefly surveys related work. Section 3 discusses the proposed service substitution framework. Section 4 provides an evaluation of our approach. Finally, Section 5 summarizes our contribution and discusses the future directions of this work.

2. RELATED WORK

Considering the importance of dealing with the independent evolution of services, the approaches that enable service substitution are quite few. In particular, the state of the art solutions can be divided in two different categories, *abstraction-based* and *adapter-based*.

As we discussed in [1], *abstraction-based* approaches enable service substitution by employing fundamental OO design principles, which are refined to the specificities of the SOA paradigm. The main idea behind these approaches is to define higher level abstractions that stand for sets of alternative concrete services and develop the client application code based on these abstractions. Specifically, in [7] and [18] the authors propose has-a abstractions, while in [16] the authors employ is-a abstractions. Going one step further, in [1] we discuss the need for a systematic reverse engineering process that extracts service abstractions out of existing services that offer similar functionality via different interfaces.

Abstraction-based approaches can not deal with the service substitution problem in cases of client applications that

[4]http://citeseer.ist.psu.edu/
[5]http://portal.acm.org/dl.cfm

do not follow the proposed methods, i.e., applications that directly access concrete services, instead of abstractions that stand for sets of alternative concrete services. Addressing this issue is the main objective of the *adapter-based* approaches. The state of the art solutions that belong in this category heavily rely on the fundamental adapter design pattern [5, 19]. Specifically, in [14] the authors discuss the issue of substituting a target service used in a client application with another concrete service, in the particular case where the interfaces of both services are derived from the same popular, or standardized interface. The client application developers are provided with resolution options for various types of incompatibilities found between the services' interfaces. Based on the developers' choices, an adapter is generated. In [13], the author relaxes the assumption that the interfaces of the target and the substitute services are derived from the same popular, or standardized interface. In particular, the proposed framework relies on a registry that maintains information about available services and adapters. Resolving incompatibilities and developing the adapters is a responsibility of the corresponding service providers. In the same spirit, in [9] the authors propose a framework that allows the semi automated generation of service adapters. To this end, the framework provides mechanisms that allow detecting both structural and protocol incompatibilities for pairs of services, involved in a substitution scenario.

The approach proposed in this paper has the same objective with *adapter-based* approaches, i.e. to enable service substitution in client applications that directly access concrete services. However, our focus of attention is on reducing the complexity of the substitution process, which, in the current state of the art, scales up with the number of available services. To achieve our goal, we borrow ideas from *abstraction-based* approaches. Specifically, we propose two substitution relations and theorems, which are used for organizing groups of related services, based on well-known consistency criteria. The service substitution process relies on these groups and thus its complexity scales up with the number of available groups, instead of scaling with the number of available services.

3. SUBSTITUTION FRAMEWORK

The main thrust of our contribution is a framework that facilitates the following tasks: (1) Systematic recovery of substitution relations between available services that may get involved in a service substitution scenario. (2) Systematic retrieval of candidate substitute services that may substitute a particular target service. (3) Systematic generation of adapters, which actually enable service substitution without changes in a client application code.

A conceptual model of the proposed framework is given in Figure 3. The *service substitution relations manager* (S2RM) recovers substitution relations, regarding available services that are progressively registered in the framework and serves as a registry that manages this information. S2RM further enables retrieving candidate services that may substitute a given target service in a particular service substitution scenario. The *service substitution adaptation manager* (S2AM) is responsible for generating adapters.

Without loss of generality we assume that services follow the W3C standard services architecture. According to this standard, a service provides an *interface* (i.e. a PortType) which consists of a set of operations (Figure 4). An *operation* corresponds to a particular service functionality, whose execution requires at most one *input message* and provides as a result at most one *output message*. An input/output message may consist of a set of distinct *parts*, each one of which is characterized by a particular XML type.

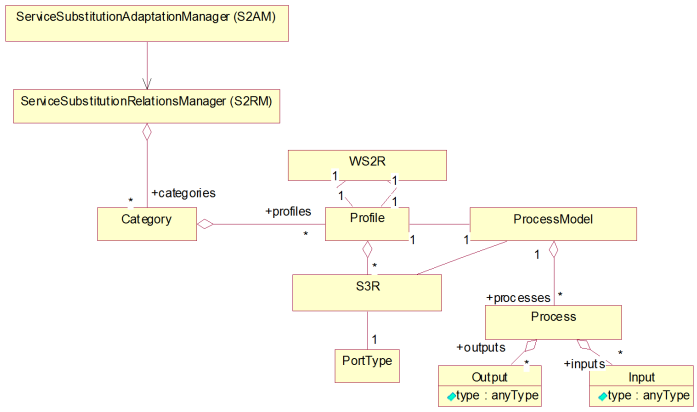


Figure 3: Conceptual model of the substitution framework.

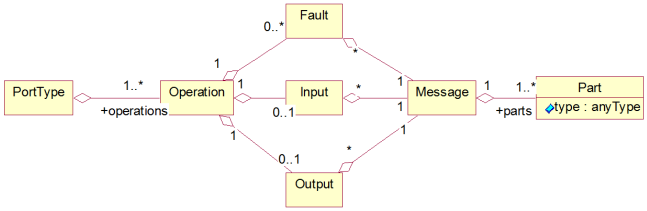


Figure 4: Conceptual model of services.

Inspired by various semantic service description languages and frameworks (e.g. OWL-S[6], SWSO[7], WSMO[8], SAWSDL[9] and WSDL-S[10]), we assume that the framework's registry is organized into different categories (e.g. publication search engines category). Each *category* comprises a set of service profiles. A service *profile* is characterized by a *process model*, which consists of a set of processes. A *process* corresponds to a functionality, which is characterized by a set of *input elements*, required for its execution, and a set of provided *output elements*. An input/output element is characterized by a particular XML data type. The service profile comprises a set of *strong service substitution relations* (S3Rs). Each S3R maps the operations of the interface of an available service to the processes of the process model that characterizes the service profile. In a sense, the main purpose of S3Rs is to associate, via a common profile, *groups of services* for which substitution involves simple renaming of operations and restructuring of the constituent parts of input and output messages. S3Rs shall be used for generating

[6]http://www.w3.org/Submission/OWL-S/
[7]http://www.daml.org/services/swsf/1.0/swso/
[8]http://www.wsmo.org/
[9]http://www.w3.org/2002/ws/sawsdl/spec/
[10]http://www.w3.org/Submission/WSDL-S/

service adapters for pairs of services that are related with a common service profile.

Moreover, different service profiles that belong in the same category may be related with *weak service substitution relations* (WS2Rs). Each WS2R maps processes of a service profile to processes of another service profile that requires fewer and/or more generic inputs to produce more and/or more specific outputs. The terms generic and specific are used here to refer to the particular data types of the inputs/outputs. WS2Rs shall allow generating adapters between pairs of services that are S3R-related with different WS2R-related profiles.

Finally, it should be noted that in existing semantic service description languages the substitution relations between a service profile and available services are referred to as groundings. Nevertheless, the actual semantics of the notion of groundings are not precisely defined for sake of flexibility. In this paper, we go one step further by defining the precise semantics of strong and weak service substitution relations, so as to guarantee a certain level of consistency when substituting a service with another that provides the same functionality via a different interface. To this end, we rely on the well known Liskov Substitution Principle (LSP) [6].

3.1 Substitution Relations & Theorems

LSP defines basic correctness rules, which guarantee that instances of a type T used in a particular software can be substituted by instances of another type S. Clearly, certain of these rules cannot be verified for services. Specifically, the invariants and the history rules [6] refer to constraints on the states and state transitions of T that must be preserved by S. In the SOA paradigm, the descriptions of services do not reveal information concerning the services states. Therefore, we do not consider these rules. Concerning the LSP pre-conditions and post-conditions rules [6], a first observation is that certain service description languages provide means for specifying pre-conditions and post-conditions. However, our experience with several collections of available services, collected by crawling the Web, shows that pre-conditions and post-conditions are rarely provided. Hence, in the definitions of our substitution relations we mainly consider the LSP contra-variance and co-variance rules, which are briefly given below.

- *Contra-variance rule for the inputs*: every operation op_S of S is mapped to an operation op_T of T that has the same number of inputs; the type of each input of op_T is a subtype of the type of the corresponding input of op_S.

- *Co-variance rule for the outputs*: both op_S, op_T have outputs, or neither has; In the former case, the types of the outputs of op_S are subtypes of the types of the corresponding outputs of op_T.

Based on the previous discussion, hereafter we define the proposed substitution relations. Naturally, the proposed relations rely on equivalence (denoted by $\equiv$) and subtyping (denoted by $\prec_{subtypeof}$) relations between the XML types involved in the specification of services interfaces and profiles. Several efficient algorithms/mechanisms that have been proposed in the past can be employed for reasoning about equivalence and subtyping relations between XML types (e.g. [3, 4, 17]). To keep our approach generic, we do not make any assumptions on a particular mechanism. As opposed to this, we assume that the proposed framework relies on an extensible set of XML types and corresponding equivalence and subtyping relations. Initially, this set comprises the standard XML data types hierarchy, where subtyping is realized based on the standard XML extension and restriction mechanisms. The set can be populated progressively with further equivalence and subtyping relations, during the registration of available services (Section 3.2) and the substitution scenarios that take place (Section 3.3). Deriving these relations, inevitably involves human effort, along with the use of a reasoning mechanism (e.g. [3, 4, 17]). The involved effort can be reduced based on the relations and theorems that are given below, since the required reasoning is performed between a target service and profiles, representing groups of services, instead of the being performed between the target service and each one of the members of these groups.

Table 1: Strong Service Substitution Relation (S3R).

$$
\begin{array}{l}
S \rightarrow_{S3R} P \Rightarrow \\
\quad \exists(M_{Op_{S,P}} : S.operations \rightarrow P.ProcessModel.processes)| \\
\quad \forall op \in S.operations|(M_{Op_{S,P}}(op) = ap) \Rightarrow \\
\quad \exists(M_{In_{S,P}} : op.Input.Message.parts \rightarrow ap.inputs, \\
\qquad M_{Out_{S,P}} : op.Output.Message.parts \rightarrow ap.outputs)| \\
\qquad \forall(i_{op} \in op.Input.Message.parts, \\
\qquad\quad o_{op} \in op.Output.Message.parts)| \\
\qquad\quad (M_{In_{S,P}}(i_{op}) = i_{ap}) \Rightarrow (i_{op}.type \equiv i_{ap}.type) \wedge \\
\qquad\quad (M_{Out_{S,P}}(o_{op}) = o_{ap}) \Rightarrow (o_{op}.type \equiv o_{ap}.type)
\end{array}
$$

The notation used in the remainder is based on the conceptual models given in Figures 3,4.

Definition 1. Strong Service Substitution Relation (S3R): A service $S : PortType$ is S3R-related with a service profile $P : Profile$ if there exist *one-to-one* and *onto* mappings between: (1) the operations of S and the processes of P, (2) the input message parts of each operation of S and the inputs of the operation's corresponding process of P and (3) the output message parts of each operation of S and the outputs of the operation's corresponding process of P, such that the types of the mapped data are equivalent. The detailed formal definition of S3R is given in Table 1, where $M_{Op_{S,P}}$, $M_{In_{S,P}}$ and $M_{Out_{S,P}}$ refer to the mappings of operations, inputs and outputs, respectively.

Concerning the LSP principle, we can prove that any two services that are S3R-related with the same profile may serve as candidate substitutes for each other because the input/output messages of their operations are equivalent. In general, message equivalence is stronger than the co-variance and contra-variance rules of LSP. More formally, the following theorem holds.

THEOREM 1. *For a pair of services $S_i, S_j : PortType$ and a profile $P : Profile$ such that $(S_i \rightarrow_{S3R} P) \wedge (S_j \rightarrow_{S3R} P)$ there exist* one-to-one *and* onto *mappings between: (1) the operations of S_i, S_j, (2) the input message parts of the mapped operations and (3) the output message parts of the mapped operations, such that the types of the mapped message parts are equivalent.*

PROOF. The mappings can be constructed by synthesizing one-to-one and onto mappings derived from the $S_i \rightarrow_{S3R} P$ relation with inverse mappings derived from the $S_j \rightarrow_{S3R} P$ relation (see Appendix A for further details).

Specifically, The synthesis, $M^{-1}_{Op_{S_j,P}} \circ M_{Op_{S_i,P}}$, is a one-to-one and onto mapping between the operations of S_i and S_j. For each pair of mapped operations $op_i \in S_i$ and $op_j = M^{-1}_{Op_{S_j,P}} \circ M_{Op_{S_i,P}}(op_i)$, the synthesis $M^{-1}_{In_{S_j,P}} \circ M_{In_{S_i,P}}$ is a one-to-one and onto mapping between the operations' input message parts, such that the types of the mapped elements are equivalent. Similarly, $M^{-1}_{Out_{S_j,P}} \circ M_{Out_{S_i,P}}$ is a one-to-one and onto mapping that maps every output message part o_{op_i} of the op_i operation of S_i, into a corresponding output message part o_{op_j} of the op_j operation of S_j, such that their types are equivalent. □

Hence, all that is needed to enable the substitution of two services that are S3R-related with the same profile is to develop an adapter that realizes the mappings specified in Theorem 1 between (1) the services' operations, (2) the input message parts of the mapped operations and (3) the output message parts of the mapped operations. Such adapters are called hereafter S3R-adapters and their precise semantics are detailed in Section 3.4.

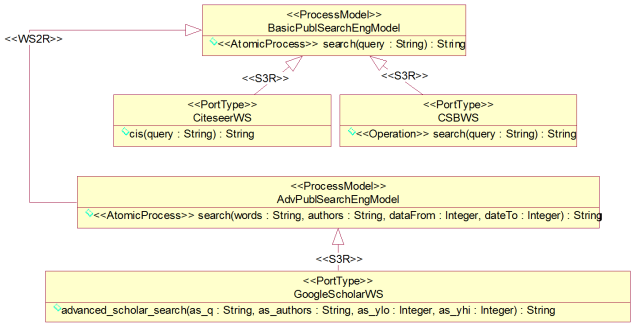


Figure 5: Examples of S3R and WS2R relations.

Taking our example, we can easily observe that the simple Citeseer service of Figure 1 is S3R-related with a basic profile for publications search engines, named `BasicPublSearchEng`, which is given in Figure 5. Specifically, the `cis` operation of the service can be mapped to the `search` process. Similarly, in Figure 5, another simple publications search service inspired by CSBib[11] is also S3R-related with the `BasicPublSearchEng` profile. Based on Theorem 1, these two services may serve as candidate substitutes for each other; this becomes evident by observing that the input and output message parts of the two services are equivalent since all of them are of the basic XML string type.

Definition 2. Weak Service Substitution Relation (WS2R): A service profile P_T is WS2R-related with a service profile P_S if there exist *one-to-one* (not necessarily *onto*) mappings between (1) the processes of P_T and P_S, (2) the inputs of P_S and P_T and (3) the outputs of P_T and P_S, such that the types of any two mapped inputs/outputs are equivalent, or compliant with the LSP contra-variance/covariance rules. The detailed definition of WS2R is given in Table 2, where $M_{Proc_{P_T,P_S}}$, $M_{In_{P_S,P_T}}$ and $M_{Out_{P_T,P_S}}$ refer to the mappings of processes, inputs and outputs, respectively.

[11] http://liinwww.ira.uka.de/bibliography/Misc/index.html

Table 2: Weak Service Substitution Relation (WS2R).

$$
\begin{array}{l}
P_T \rightarrow_{WS2R} P_S \Rightarrow \\
\quad \exists (M_{Proc_{P_T,P_S}} : P_T.ProcessModel.processes \rightarrow \\
\qquad\qquad P_S.ProcessModel.processes) | \\
\quad \forall ap_T \in P_T.ProcessModel.processes | \\
\quad\quad (M_{Proc_{P_T,P_S}}(ap_T) = ap_S) \Rightarrow \\
\quad\quad\quad \exists (M_{In_{P_S,P_T}} : ap_S.inputs \rightarrow ap_T.inputs, \\
\quad\quad\quad\quad M_{Out_{P_T,P_S}} : ap_T.outputs \rightarrow ap_S.outputs) | \\
\quad\quad\quad\quad \forall (i_{ap_S} \in ap_S.inputs, o_{ap_T} \in ap_T.outputs) | \\
\quad\quad\quad\quad\quad ((M_{In_{P_S,P_T}}(i_{ap_S}) = i_{ap_T}) \Rightarrow \\
\quad\quad\quad\quad\quad\quad ((i_{ap_T}.type \equiv i_{ap_S}.type) \vee \\
\quad\quad\quad\quad\quad\quad\quad (i_{ap_T}.type \prec_{subtypeof} i_{ap_S}.type))) \\
\quad\quad\quad\quad\quad \wedge \\
\quad\quad\quad\quad\quad ((M_{Out_{P_T,P_S}}(o_{ap_T}) = o_{ap_S}) \Rightarrow \\
\quad\quad\quad\quad\quad\quad ((o_{ap_S}.type \equiv o_{ap_T}.type) \vee \\
\quad\quad\quad\quad\quad\quad\quad (o_{ap_S}.type \prec_{subtypeof} o_{ap_T}.type)))
\end{array}
$$

Roughly, the fact that the mapping between the inputs of P_S and P_T is not necessarily onto implies that P_S may have fewer inputs than P_T. On the other hand, since the mapping between the outputs of P_T and P_S is not necessarily onto, P_S may have more outputs than P_T. Moreover, the inputs of P_S may be of a more generic type, while the outputs of P_S may be of a more concrete type.

Regarding the LSP principle, we can prove that if two services S_T, S_S are S3R-related with two WS2R-related profiles P_T and P_S, respectively, then S_S may serve as a candidate substitute of S_T. Formally, the following theorem holds.

THEOREM 2. *For any two services S_T, S_S and profiles P_T, P_S such that,*

$$(S_T \rightarrow_{S3R} P_T) \wedge (S_S \rightarrow_{S3R} P_S) \wedge (P_T \rightarrow_{WS2R} P_S)$$

there exist one-to-one *mappings between (1) the operations of S_T and S_S, (2) the input message parts of the mapped operations and (3) the output message parts of the mapped operations, such that the types of any two mapped input/output message parts are equivalent, or compliant with the LSP contra-variance/covariance rules, respectively.*

PROOF. The mappings can be constructed by synthesizing one-to-one and onto mappings derived from the $S_T \rightarrow_{S3R} P_T$ relation with one-to-one mappings derived from the $P_T \rightarrow_{WS2R} P_S$ and inverse mappings derived from the $S_S \rightarrow_{S3R} P_S$ relation (see Appendix B for further details).

Specifically, the synthesis $M^{-1}_{Op_{S_S,P_S}} \circ M_{Proc_{P_T,P_S}} \circ M_{Op_{S_T,P_T}}$, is a one-to-one mapping between the operations of S_T and the operations of S_S. For each pair of mapped operations op_T and $op_S = M^{-1}_{Op_{S_S,P_S}} \circ M_{Proc_{P_T,P_S}} \circ M_{Op_{S_T,P_T}}(op_T)$, the synthesis $M^{-1}_{In_{S_S,P_S}} \circ MRes^{-1}_{In_{P_S,P_T}} \circ M_{In_{S_T,P_T}}$ is a one-to-one mapping between the operations' input message parts, such that the types of the mapped elements are either equivalent, or follow the LSP contra-variance rule. In this synthesis, $MRes_{In_{P_S,P_T}}$ denotes the one-to-one and onto mapping derived from $M_{In_{P_S,P_T}}$, by the restriction of the codomain of $M_{In_{P_S,P_T}}$ to the range of $M_{In_{P_S,P_T}}$ (see Appendix B for further details).

Finally, the synthesis $M^{-1}_{Out_{S_S,P_S}} \circ M_{Out_{P_T,P_S}} \circ M_{Out_{S_T,P_T}}$, is a one-to-one mapping between the output message parts of each pair of mapped operations, such that the types of the mapped elements are either equivalent, or follow the LSP co-variance rule. □

Hence, substituting S_T with S_S amounts to developing a corresponding adapter with respect to the mappings discussed in Theorem 2 and its proof. Hereafter, we call such adapters WS2R-adapters and their semantics are detailed in Section 3.4.

Returning to our example, suppose that the GoogleScholar service is S3R-related with an advanced profile for publications search engines, named `AdvPublSearchEng` (Figure 5). This profile is WS2R-related with the `BasicPublSearchEng` profile that was previously discussed. Specifically, the `search` process of the `AdvPublSearchEng` profile is mapped to the `search` process of the `BasicPublSearchEng` profile. Moreover, the `query` input of the `BasicPublSearchEng search` process can be mapped to the `words` input, or to the `authors` input of the `AdvPublSearchEng search` process, since their types are equivalent (choosing between alterative mappings, derived by the framework, is up to the users in charge of inspecting the mappings (Section 3.2, 3.3)). Similarly, the types of the outputs of both search processes are equivalent. Based on Theorem 2, we can conclude that Citeseer or CSBib may serve as candidate substitutes of the GoogleScholar service. The previous becomes evident by observing that the `advanced_scholar_search` operation of GoogleScholar can be mapped, via the WS2R and S3R relations, to the `cis` operation of Citeseer, or to the `search` operation of CSBib.

3.2 Recovering Service Substitution Relations

To register a new service in the registry managed by the S2RM component (Figure 3), the service provider has to choose an already existing category of services, or create a new one in collaboration with the framework administrator. In the former case, the registration of the new service triggers the need for mining S3R relations amongst the new service and the profiles that constitute the selected category. In the absence of such relations, a newly created profile is inserted in the selected category. The new profile is generated with respect to the interface of the new service. Moreover, the new service profile comprises a S3R relation that associates it with the registered service. Finally, the insertion of the new profile in the given category further involves the potential of establishing appropriate WS2R relations between the new profile and previously existing ones.

An abstract view of the algorithm that actually realizes the service registration process is given in Table 3. Given a service, `s`, and a category, `c`, the algorithm generates a profile, `newP`, that corresponds to the interface of `s`. Then, the algorithm iterates over the set of available profiles `c.profiles` that belong to `c`:

1. For every profile `p` of `c.profiles`, the algorithm examines (with respect to Definition 1) whether a S3R relation can be established between `s` and `p` (Table 3, lines 5-8):
   (a) If the previous is possible (i.e. `checkS3R(s, p)` returns true) corresponding, $M_{Op_{s,p}}$, $M_{In_{s,p}}$, $M_{Out_{s,p}}$ mappings are generated. The recovered S3R relation and the mappings are inspected and validated by the service provider. If the validation is positive the new S3R relation is inserted in the framework and the algorithm terminates.
   (b) Otherwise, `newP` is examined against `p` (Table 3, lines 9-12). If `newP` can be WS2R-related with `p` (i.e. `checkWS2R(newP, p)` returns true), the latter is inserted in a set of candidate profiles and corresponding $M_{Proc_{newP,p}}$, $M_{In_{p,newP}}$, $M_{Out_{newP,p}}$ mappings are generated.
2. At the end of this process, if `s` was not S3R-related with any of the profiles that belong to `c`, WS2R-relations are established between `newP` and the profiles of the candidate set (Table 3, lines 19-21). Each established relation and mappings are inspected and validated by the service provider.

Table 3: Mining service substitution relations.

Algorithm:	RecoverSubstRelations
Input:	s:PortType, c:Category
Output:	c:Category

```
1  var newP:Profile
2  var candidateParents: setof Profile
3  newP = createProfile(s)
4  forall (p:Profile in c.profiles)
5    if ( checkS3R(s, p) == TRUE ) then
6      p.S3R = p.S3R ⋃ {s --S3R--> p}
7      return c
8    endif
9    if ( checkWS2R(newP, p) == TRUE ) then
10     candidateParents = candidateParents ⋃ {p}
11   endif
12 endforall
13 forall (pruned:Profile in candidateParents)
14   if (exists p in candidateParents |{p --WS2R--> pruned} in pruned.WS2R) then
15     candidateParents = candidateParents - {pruned}
16   endif
17 endforall
18 c.profiles = c.profiles ⋃ newP
19 forall (p:Profile in candidateParents)
20   p.WS2R = p.WS2R ⋃ {newP --WS2R--> p}
21 endforall
22 return c
```

In our example, suppose that the search engines category of the framework is initially empty. Then, Figure 5 gives the result of registering the CSBib service, followed by Citeseer and GoogleScholar. Initially, the registration of CSBib results in the creation of the `BasicPublSearchEng` profile, which is generated based on the `CSBibWS` interface (naming the profile and manipulating further documentation details is assisted by the service provider). According to Definition 1, Citeseer is found S3R-related with `BasicPublSearchEng`. Hence, the registration of Citeseer results in the insertion of the new S3R relation that associates it with `BasicPublSearchEng`. Finally, according to Definition 1, GoogleScholar and `BasicPublSearchEng` are not be S3R-related. However, based on Definition 2, the `AdvPublSearchEng` profile (generated based on the `GoogleScholarWS` interface) is found WS2R-related with `BasicPublSearchEng`. Therefore, during the registration of GoogleScholar the aforementioned WS2R relation is established.

3.3 Retrieving Substitute Services

The retrieval of services that may substitute a target service used in a client software is also realized by the S2RM component. The specific algorithm used is given in Table 4. In detail, the client application developer provides as input the `target` service and a selected category, `c`, that may contain information about relevant services. Based on the

Table 4: Retrieval of substitute services.

```
Algorithm:  RetrieveSubstService
Input:      target:PortType, c:Category
Output:     adaptees: setof PortType
1  var targetP:Profile
2  var candidateProfiles: setof Profile
3  var adapteeProfile:Profile
4  targetP = createProfile(target)
5  forall (p:Profile in c.profiles)
6    if ( checkS3R(target, p) == TRUE ) then
7      adaptees = selectService(p.S3R)
8      return adaptee
9    endif
10   if ( checkWS2R(targetP, p) == TRUE ) then
11     candidateProfiles = candidateProfiles ∪ {p}
12
13   endif
14 endforall
15 forall (pruned:Profile in candidateProfiles)
16   if (exists p in candidateProfiles|{p --WS2R--> pruned} in pruned.WS2R) then
17     candidateProfiles = candidateProfiles - {pruned}
18   endif
19 endforall
20 adapteeProfile = selectProfile(candidateProfiles)
21 adaptees = selectService(adapteeProfile.S3R)
22 return adaptees
```

interface of the target service, a corresponding profile, `targetP`, is generated in a straightforward way. Following, the algorithm iterates over the set of profiles that belong to `c`.

1. For every profile, `p` of `c.profiles`, it is checked whether the `target` service is S3R-related with `p` (Table 4, lines 6-9).
   (a) If the previous holds (i.e. `checkS3R(target, p)` returns true), $M_{Op_{target,p}}$, $M_{In_{target,p}}$, $M_{Out_{target,p}}$ mappings are calculated. According to Theorem 1 we have that any service, `adaptee`, that was previously registered to the framework and was found S3R-related with `p` (Section 3.2 Table 3) may serve as a candidate substitute of `target`. Therefore, a set of such services is selected by inspecting the S3R relations of `p` (Table 4, line 7) and the algorithm terminates.
   (b) Otherwise, the algorithm checks whether the profile `targetP` that was generated based on the `target` service is WS2R-related with `p` (Table 4, lines 10-14). If this holds (i.e. `checkWS2R(targetP, p)` returns true), `p` is inserted in a set of candidate profiles. According to Theorem 2, the candidate profiles are S3R-related with services that may be used as substitutes of `target`; these services require fewer and/or more generic inputs to produce more and/or more specific outputs than the `target` service. Moreover, $M_{Proc_{targetP,p}}$, $M_{In_{p,targetP}}$, $M_{Out_{targetP,p}}$ mappings are calculated.
2. At the end of this process, if `target` was not S3R-related with any of the profiles that belong in `c` the set of candidate profiles is explored. As previously mentioned, the services which are S3R-related with the profiles that belong in this set may serve as candidate substitutes of `target`. Therefore, the algorithm proceeds as follows: The set of candidate profiles is inspected by the client application developer and a particular profile, `adapteeProfile`, is selected; then, the S3R relations between available services and the `adapteeProfile` are inspected, towards selecting a set of such services that may substitute `target`.

In our example, assume that the `target` service is GoogleScholar. Given the situation established in Figure 5, the algorithm shall find (based on Definition 2) that the profile, generated from the `GoogleScholarWS` interface, is WS2R-related with the `BasicPublSearchEng` profile. Then, according to Theorem 2, the services that are S3R-related with this profile (i.e. Citeseer, CSBib) may be selected, towards substituting GoogleScholar in the client code.

3.4 Generating adapters

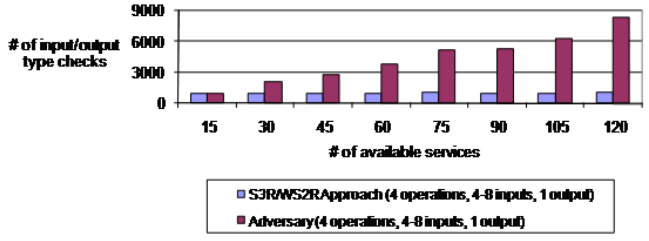
Substituting a `target` service with a substitute service, `adaptee`, retrieved according to the process discussed in Section 3.3 consists of generating an adapter that maps invocations of operations, offered by the `target` service interface, into invocations of corresponding operations, provided by the `adaptee` service interface. Technically, the generated adapters are also W3C services. The mappings of operations and input/output message parts (Section 3.2, 3.3) are given to the S2AM component (Figure 3); the generated adapter code that realizes the mappings of input/output message parts may involve: (1) simple type casting operations, if the types of the message parts are standard XML types, or (2) more complex conversions, if the types of the message parts are user-defined complex XML types.

```
1 class GoogleScholar2CiteseerAdaptor {
2   CiteseerWSInterface ref;
3   //……….
4   String advanced_scholar_search (
5    String as_q, String as_authors,
6    int as_ylo, int as_yhi) {
7     String publications = ref.cis(as_q);
8     return publications;
9 } }
```

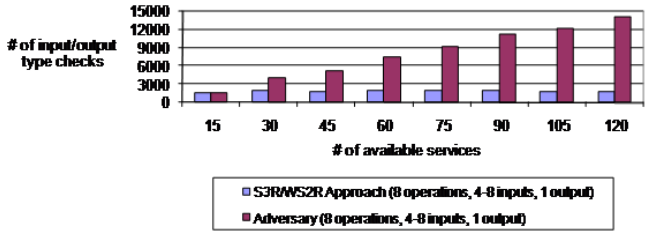
Figure 6: Example WS2R-adapter for GoogleScholar and Citeseer.

Specifically, if the `target` and the `adaptee` services are S3R-related with the same profile, P, then a S3R-adapter is generated. In particular, for every operation, *op*, provided by the `target` interface, a homonymous operation is generated for the S3R-adapter. According to Theorem 1, the implementation of this operation invokes the $M^{-1}_{Op_{adaptee,P}} \circ M_{Op_{target,P}}(op)$ operation, offered by the `adaptee` service interface. As discussed in Section 3.3, the $M_{Op_{target,P}}$ mapping is derived during the retrieval process. On the other hand, the $M_{Op_{adaptee,P}}$ mapping is derived during `adaptee` service registration process detailed in Section 3.2. The input and output message parts of *op* are transformed with respect to the $M^{-1}_{In_{adaptee,P}} \circ M_{In_{target,P}}$ and the $M^{-1}_{Out_{adaptee,P}} \circ M_{Out_{target,P}}$ mappings, discussed in Theorem 1.

If the `target` and the `adaptee` services are S3R-related with two different WS2R-related profiles, $targetP$ and P, respectively a WS2R-adapter is generated. For every operation, *op*, provided by the `target` interface, a homonymous operation is generated for the WS2R-adapter. According to Theorem 2, the implementation of *op* invokes the $M^{-1}_{Op_{adaptee,P}} \circ M_{Proc_{targetP,P}} \circ M_{Op_{target,targetP}}(op)$ operation, offered by the `adaptee` interface. Moreover, the input and output message parts of *op* are transformed according to the $M^{-1}_{In_{adaptee,P}} \circ MRes^{-1}_{In_{P,targetP}} \circ M_{In_{target,targetP}}$ and

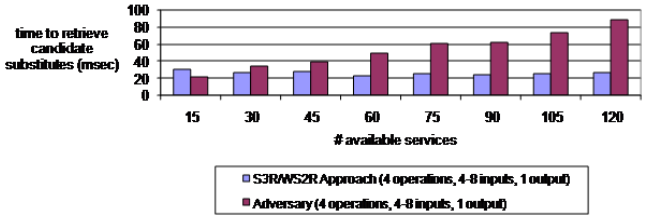


(a) 1st variant.

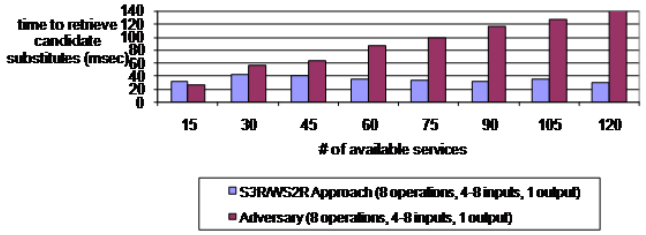


(b) 2nd variant.

Figure 7: 1st Set of experiments.



(a) 1st variant.



(b) 2nd variant.

Figure 8: 2nd Set of experiments.

the $M^{-1}_{Out_{adaptee,P}} \circ M_{Out_{targetP,P}} \circ M_{Out_{target,targetP}}$ mappings, discussed in Theorem 2.

In our example, the adapter of Figure 6 is generated in the case where the `target` service is GoogleScholar and the retrieved `adaptee` is Citeseer (Section 3.3); its realization is based on the mappings discussed in Section 3.1. Briefly, the implementation of the `advanced_scholar_search` operation invokes the `cis` operation of the Citeseer service. The `as_q` input message part of the `advanced_scholar_search` operation is mapped to the `query` input message part of the `cis` operation, while the remaining input message parts (i.e. `as_authors`, `as_ylo`, `as_yhi`) of `advanced_scholar_search` are ignored.

4. EVALUATION

The main objective of the proposed approach is to reduce the complexity of service substitution in client applications that use concrete services. To assess the proposed approach with respect to this objective we performed two sets of experiments. In both sets we compared the proposed approach, against a typical adversary that does not rely on the substitution relations and theorems discussed in Section 3. The adversary assumes a registry of available services and tries to retrieve all possible candidate substitutes services for a given target service. The basic criterion for retrieving a candidate substitute is that there exist one-to-one and onto mappings between the operations, the input message parts and the output message parts of the target and the substitute service, such that the types of the mapped elements are equivalent.

In the first set of experiments, our goal was to compare *the effort required for the retrieval of candidate substitute services, in the case where this task involves human intervention.* A main indication of this effort (Section 3.1) is the comparisons required for the discovery of equivalence and subtyping relations, between user-defined input/output data types, used in the services involved in our substitution scenarios. In the second set of experiments, our goal was to compare *the time required for the retrieval of candidate substitute services, in the case where this task is fully automated* (the client application developers are involved only to inspect the results of the retrieval (Section 3.3)). Therefore, in this set of experiments the services involved in the substitution scenarios were using input/output data of standard XML types.

In both sets of experiments we assumed 8 different configurations where the cardinality of available services in the registries varied in the range [15, 120]. Moreover, for each configuration we assumed 2 different variants of services. In the first variant, the available services offered up to 4 operations. In the second variant, the available services offered up to 8 operations. The operations had one output message part and the number of input message parts varied in the range [4,8]. The services were generated by randomly selecting data types with a uniform distribution, from corresponding hierarchies that we developed for the purpose of our experiments. In all cases, the generation process was such that the services could be organized in 15 groups, characterized by corresponding profiles and S3R relations. The cardinality of WS2R relations between profiles ranged up to 4. In both sets of experiments we randomly generated 500 target services, which served as input to the proposed approach and the adversary. The experiments were performed on a P-IV 1.67GHz, 2GB RAM.

Figure 7 summarizes the results from the 1st set of experiments. In particular, the mean (over the 500 target services that were used) number of input/output type checks performed to retrieve candidate substitute services is given for the proposed approach and the adversary. The benefits of the proposed approach in terms of the effort required for retrieving candidate substitute services become evident, as the cardinality of available services increased. In the case of the proposed approach, the required mean number of input/output type checks remained quite stable, since the number of profiles that group available services was stable in our experimental setup, while in the case of the adversary the required number of type checks scaled up with the number of services. Similarly, Figure 8 gives the results of the 2nd set of experiments. The mean (over the 500 target ser-

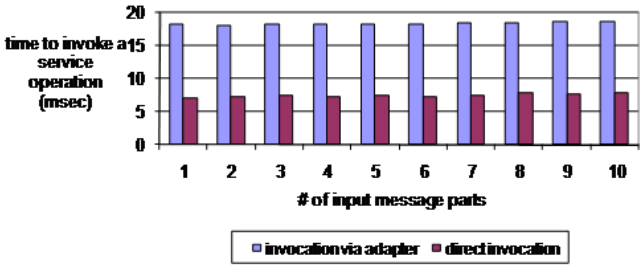


Figure 9: 3rd Set of experiments.

vices) time required for the retrieval of candidate substitute services in the proposed approach remained stable, while in the adversary it scaled up with the number of available services.

Finally, given that the proposed approach concerns the generation of adapters, we further performed a third set of experiments to evaluate the overhead introduced by the adapters in the execution of client applications. Specifically, we compared the time required for accessing a service operation via a generated adapter, against the time required for accessing the same operation directly. We assumed 10 different cases, where we varied the number of input message parts of the invoked operations in the range [1,10]; the parts were of standard XML types. To give a clearer view of the adapter overhead, the invoked operations were trivial, in the sense that they immediately returned a default result, without further processing. The invoking application was executed on a P-IV, 1.67 GHz, 2 GB RAM, while the adapter and the service were deployed on another P-IV, 1.67 GHz, 2 GB RAM. The machines were connected via a typical 100Mbps LAN. As expected, the use of the adapter was quite expensive (Figure 9). The number of input message parts did not affect the adapter overhead, since the execution cost of the generated code that maps the message parts of the target service to the ones of the substitute service is quite small, compared to the cost for invoking 2 service operations, instead of one.

5. CONCLUSION

In this paper we proposed a framework that deals with the complexity of the service substitution problem. The proposed framework relies on a formal foundation that allows organizing available services into groups. Then, the complexity of service substitution scales up with the number of available groups, instead of scaling up with the number of available services. Indeed, our experimental results highlighted the aforementioned benefit.

Currently, we plan to further evaluate our approach based on real collections of services that were found by crawling the Web. Moreover, we consider extending the proposed approach to account for protocol compatibility issues [9]. Finally, we work towards a reverse engineering process that would allow to improve the organization of services into groups, by recovering service abstractions from a set of available services [1].

6. REFERENCES

[1] D. Athanasopoulos, A. Zarras, and V. Issarny. Towards the Maintenance of Service Oriented Software. In *Proceedings of the 3rd CSMR Workshop on Software Quality and Maintenance (SQM'09)*, 2009.

[2] J. Cardoso and A. Sheth. *Semantic Web Services, Processes and Applications*. Springer, 2006.

[3] S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[4] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE'02)*, 2002.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[6] B. Liskov and J. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems (ACM TOPLAS)*, 16(6):1811–1841, 1994.

[7] L. Melloul and A. Fox. Reusable Functional Composition Patterns for Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 2004.

[8] S. B. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient Semantic Service Discovery in Pervasive Computing Environments. In *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE'06)*, pages 240–259, 2006.

[9] H. R. M. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi Automated Adaptation of Service Interactions. In *Proceedings of the International World Wide Web Conference (WWW'07)*, 2007.

[10] E. D. Nitto, M. D. Penta, A. Gambi, G. Ripa, and M. Villani. Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ISOC'07)*, 2007.

[11] S. Oundhakar, K. Verma., K.Sivashanugam, A. Sheth, and J. Miller. Discovery of Web Services in a Multi-Ontology and Federated Registry Environment. *International Journal of Web Services Research*, 1(3):1–32, 2005.

[12] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, 2002.

[13] S. R. Ponnekanti. Application-Service Interoperation Without Standardized Service Interfaces. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2003.

[14] S. R. Ponnekanti and A. Fox. Interoperability Among Independently Evolving Web Services. In *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE)*, 2004.

[15] F. Raimondi, J. Skene, and W. Emmerich. Efficient Online Monitoring of Web Service SLAs. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*

(FSE'08), 2008.

[16] Y. Taher, D. Benslimane, M.-C. Fauvet, and Z. Maamar. Towards an Approach for Web Services Substitution. In *Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS)*, 2006.

[17] Y. Wang, D. DeWitt, and J. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*, 2003.

[18] J. Yang and M. Papazoglou. Service Components for Managing the Lifecycle of Service Compositions. *Information Systems*, 29(2):97–125, 2004.

[19] D. Yellin and R. Strom. Protocol Specification and Component Adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.

APPENDIX

A. PROOF OF THEOREM 1

Theorem 1 can be deduced from the definition of the S3R relation as follows: The $S_i \rightarrow_{S3R} P$ relation implies that there exists a one-to-one and onto mapping $M_{Op_{S_i,P}}$ between the operations of S_i and the processes of P. Moreover, for every pair of mapped operation, op_i, and process, $ap = M_{Op_{S_i,P}}(op_i)$, there exists a one-to-one and onto mapping, $M_{In_{S_i,P}}$, between the operation's input message parts and the process's inputs, such that the types of a mapped pair of elements, i_{op_i}, i_{ap}, are equivalent. Similarly, the $S_j \rightarrow_{S3R} P$ relation implies that there exists a one-to-one and onto mapping $M_{Op_{S_j,P}}$ between the operations of S_j and the processes of P. For each pair of mapped operation, op_j, and process, $ap = M_{Op_{S_j,P}}(op_j)$, there exists a one-to-one and onto mapping, $M_{In_{S_j,P}}$, between the operation's input message parts and the process's inputs such that the types of the mapped elements are equivalent. The inverse of these mappings are also one-to-one and onto.

Based on the above we have that the synthesis, $M^{-1}_{Op_{S_j,P}} \circ M_{Op_{S_i,P}}$, is a mapping between the operations of S_i and S_j; this mapping is one-to-one and onto since its constituents $M^{-1}_{Op_{S_j,P}}$, $M_{Op_{S_i,P}}$ are one-to-one and onto. Moreover, for each pair of mapped operations op_i and $op_j = M^{-1}_{Op_{S_j,P}} \circ M_{Op_{S_i,P}}(op_i)$, the synthesis $M^{-1}_{In_{S_j,P}} \circ M_{In_{S_i,P}}$ is a one-to-one and onto mapping between the operations' input message parts. Finally, this synthesis of mappings preserves type equivalence between the mapped elements, since its constituent mappings preserve type equivalence.

By following similar steps, we can conclude that $M^{-1}_{Out_{S_j,P}} \circ M_{Out_{S_i,P}}$ is a one-to-one and onto mapping that maps every output message part o_{op_i} of the op_i operation of S_i, into a corresponding output message part o_{op_j} of the $op_j = M^{-1}_{Op_{S_j,P}} \circ M_{Op_{S_i,P}}(op_i)$ operation of S_j, such that their types are equivalent.

B. PROOF OF THEOREM 2

Theorem 2 can be deduced from the definitions of S3R and WS2R as follows:

The $S_T \rightarrow_{S3R} P_T$ relation implies that there exists a one-to-one and onto mapping $M_{Op_{S_T,P_T}}$ between the operations of S_T and the processes of P_T. Moreover, for every pair of mapped operation, op_T, and process, $ap_T = M_{Op_{S_T,P_T}}(op_T)$, there exists a one-to-one and onto mapping, $M_{In_{S_T,P_T}}$, between the operation's input message parts and the process's inputs such that the types of the mapped elements are equivalent. Formally:

$$\forall i_{op_T} \in op_T.Input.Message.parts| \quad (1)$$
$$(M_{In_{S_T,P_T}}(i_{op_T}) = i_{ap_T}) \Rightarrow (i_{op_T}.type \equiv i_{ap_T}.type)$$

The $P_T \rightarrow_{WS2R} P_S$ relation implies that there is a one-to-one mapping $M_{Proc_{P_T,P_S}}$ between the processes of P_T and the processes of P_S. According to this mapping, for each pair of mapped processes $ap_T \in P_T$, $ap_S \in P_S$ there is a one-to-one mapping, $M_{In_{P_S,P_T}}$, between the processes' inputs such that the types of the mapped inputs are equivalent, or the LSP contra-variance rule holds. Therefore, the following holds:

$$\forall i_{ap_S} \in ap_S.inputs| \quad (2)$$
$$(M_{In_{P_S,P_T}}(i_{ap_S}) = i_{ap_T}) \Rightarrow$$
$$((i_{ap_T}.type \equiv i_{ap_S}.type) \vee$$
$$(i_{ap_T}.type \prec_{subtypeof} i_{ap_S}.type))$$

The $S_S \rightarrow_{S3R} P_S$ relation implies that there exists a one-to-one and onto mapping $M_{Op_{S_S,P_S}}$ between the operations of S_S and the processes of P_S. Furthermore, for each pair of mapped operation and process there exists a one-to-one and onto mapping, $M_{In_{S_S,P_S}}$, between the operation's input message parts and the process's inputs such that the types of the mapped elements are equivalent. The inverse of these mappings are also one-to-one and onto. Hence we have:

$$\forall i_{ap_S} \in ap_S.inputs| \quad (3)$$
$$(M^{-1}_{In_{S_S,P_S}}(i_{ap_S}) = i_{op_S}) \Rightarrow (i_{ap_S}.type \equiv i_{op_S}.type)$$

Based on the above we have that the synthesis, $M^{-1}_{Op_{S_S,P_S}} \circ M_{Proc_{P_T,P_S}} \circ M_{Op_{S_T,P_T}}$, is a mapping between the operations of S_T and the operations of S_S. This mapping is one-to-one because its constituents, $M^{-1}_{Op_{S_S,P_S}}$, $M_{Proc_{P_T,P_S}}$ and $M_{Op_{S_T,P_T}}$ are one-to-one mappings. Further, for each pair of mapped operations op_T and $op_S = M^{-1}_{Op_{S_S,P_S}} \circ M_{Proc_{P_T,P_S}} \circ M_{Op_{S_T,P_T}}(op_T)$, the synthesis $M^{-1}_{In_{S_S,P_S}} \circ MRes^{-1}_{In_{P_S,P_T}} \circ M_{In_{S_T,P_T}}$ is a one-to-one mapping between the operations' input message parts. In this synthesis, $MRes_{In_{P_S,P_T}}$ denotes the one-to-one and onto mapping derived from $M_{In_{P_S,P_T}}$, by the restriction of the codomain of $M_{In_{P_S,P_T}}$ to the range of $M_{In_{P_S,P_T}}$. For the synthesis $M^{-1}_{In_{S_S,P_S}} \circ MRes^{-1}_{In_{P_S,P_T}} \circ M_{In_{S_T,P_T}}$, (1), (2) and (3) imply that the types of the mapped elements are either equivalent, or follow the LSP contra-variance rule. By following similar steps, we can easily conclude that the synthesis, $M^{-1}_{Out_{S_S,P_S}} \circ M_{Out_{P_T,P_S}} \circ M_{Out_{S_T,P_T}}$, is a one-to-one mapping between the output message parts of each pair of mapped S_T and S_S operations, such that the types of the mapped elements are either equivalent, or follow the LSP co-variance rule.