Service Substitution Revisited

Dionysis Athanasopoulos*†, Apostolos V. Zarras* and Valerie Issarny†
* *Dept. of Computer Science, University of Ioannina (UoI), Greece*
Email: {dathanas,zarras}@cs.uoi.gr
† *INRIA-Rocquencourt, Domaine de Voluceau, France*
Email: Valerie.Issarny@inria.fr

***Abstract*—In this paper we propose a framework that reduces the complexity of service substitution. The framework 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.**

I. Introduction

Service substitution is a key issue towards dealing with the independent evolution of services along with their variation in quality (e.g. performance, availability, reliability) [1], [2], [3]. Research efforts that focus on service substitution can be divided in two categories. 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* [4], [5], [6]. The second category comprises *adapter-based* approaches, which deal with *existing client applications that use concrete services* [7], [8], [9], [10], [11], [12], [13]. 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, which allows accessing the functionality of the substitute service through the original target interface, without modifying the client application code.

While considering *adapter-based* approaches the following issue is raised: *the effort and time required by the service substitution process scales up with the number of available services that should be examined as potential candidate substitutes of the target service*. This problem is a serious drawback towards a practical service substitution approach if we consider that the service substitution process involves human intervention to validate the mapping between target and substitute services.

In this paper, we share the objective of *adapter-based* approaches. 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) [14] (Section 2). 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 (Section 3).

The long version of this paper with detailed definitions and proofs can be found http://www.cs.uoi.gr/∼zarras/papers/C23.pdf

II. Substitution Framework

A conceptual model of the proposed framework is given in Figure 1. 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. 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. Inspired by various semantic service description languages and frameworks (e.g. OWL-S[1], SWSO[2], WSMO[3]), we assume that the framework's registry is organized into different categories. 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* and a set of

[1]http://www.w3.org/Submission/OWL-S/
[2]http://www.daml.org/services/swsf/1.0/swso/
[3]http://www.wsmo.org/

output elements. An input/output element is characterized by a particular XML data type.

A 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 are used for generating 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 allow generating adapters between pairs of services that are S3R-related with different WS2R-related profiles.

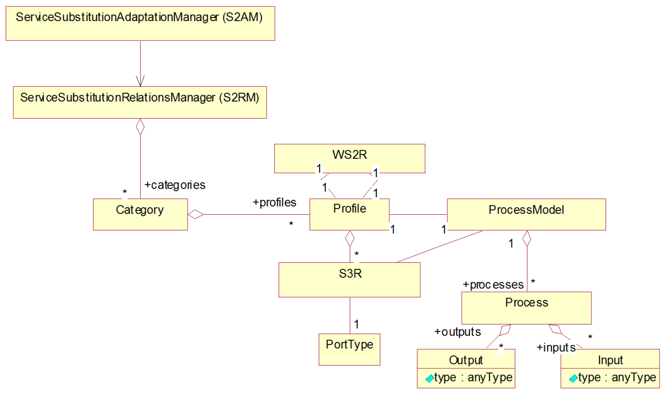


Figure 1. Conceptual model of the substitution framework.

Substitution Relations & Theorems: The Liskov Substitution Principle (LSP) [14] 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 (e.g. invariants and history rules that refer to state properties) cannot be verified for services. In the definitions of our substitution relations we mainly consider the LSP contra-variance and co-variance rules [14]. Hence, the proposed relations rely on equivalence and subtyping 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 (we specifically used [15]). However, to keep our base 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 and the substitution scenarios that take place.

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.

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 long version). ■

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.

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.

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 long version). ■

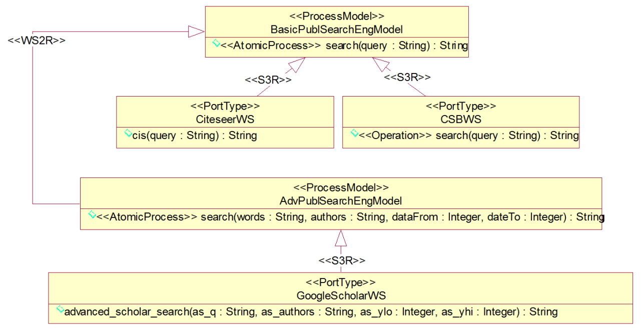

<<WS2R>>
<<ProcessModel>>
BasicPublSearchEngModel
<<AtomicProcess>> search(query : String) : String
<<S3R>>
<<S3R>>
<<PortType>>
CiteseerWS
cis(query : String) : String
<<PortType>>
CSBWS
<<Operation>> search(query : String) : String
<<ProcessModel>>
AdvPublSearchEngModel
<<AtomicProcess>> search(words : String, authors : String, dataFrom : Integer, dateTo : Integer) : String
<<S3R>>
<<PortType>>
GoogleScholarWS
advanced_scholar_search(as_q : String, as_authors : String, as_ylo : Integer, as_yhi : Integer) : String

Figure 2. Examples of S3R and WS2R relations.

Figure 2 gives examples of related profiles and services. Observe that the `CiteseerWS` service, inspired by the Citeseer[4] publications search engine, is S3R-related with a basic profile for publications search engines, named `BasicPublSearchEng`. Specifically, the `cis` operation of the service can be mapped to the `search` process. Similarly, another simple publications search service inspired by CSBib[5] 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. The `GoogleScholarWS` service, inspired by GoogleScholar[6], is S3R-related with an advanced profile for publications search engines, named `AdvPublSearchEng`. This profile is WS2R-related with the `BasicPublSearchEng` profile. 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 that validate the mappings). Similarly, the types of the outputs of both search processes are equivalent. Based on Theorem 2, we can conclude that `CiteseerWS` or `CSBibWS` may serve as candidate substitutes of the `GoogleScholarWS` service; 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.

[4]http://citeseer.ist.psu.edu/
[5]http://liinwww.ira.uka.de/bibliography/Misc/index.html
[6]http://scholar.google.com/advanced_scholar_search

Organizing Services: To register a new service in the registry managed by the S2RM component (Figure 1), 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 is followed by the establishment of S3R relations amongst the new service and the profiles that constitute the selected category and the generation of corresponding *mappings* between the service interface and the profiles processes, inputs and outputs. In the absence of S3R 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 and the generation of corresponding *mappings* (see long version for details on the service registration algorithm).

In our example, suppose that the search engines category of the framework is initially empty. Then, Figure 2 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, 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.

Retrieving Services: To retrieve a set of services that may substitute a target service used in a client software 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 interface of the target service, a corresponding profile, *targetP*, is generated in a straightforward way. Following, the retrieval procedure iterates over the set of profiles that belong to C towards locating a profile P such that $target \rightarrow_{S3R} P$, or $targetP \rightarrow_{WS2R} P$. If such a profile is found, any of the services that are S3R related with P can be used as substitutes for *target* (according to Theorems 1, 2). The retrieval procedure returns as output these services, along with the *mappings* that relate their interfaces with the interface of the *target* service (see long version for details on the service retrieval algorithm).

In our example, assume that the `target` service is GoogleScholar. Given the situation established in Figure 2, the retrieval procedure 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.

Generating Adaptors: Substituting the *target* service with a retrieved substitute service, *adaptee*, 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. The *mappings* of operations and input/output message parts are given to the S2AM component (Figure 1). Technically, the generated adapter is also a W3C service; the generated code that implements 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.

```
class GoogleScholar2CiteseerAdaptor {
  CiteseerWSInterface ref;
  //---------------.
  String advanced_scholar_search (
   String as_q, String as_authors,
   int as_ylo, int as_yhi) {
    String publications = ref.cis(as_q);
    return publications;
 } }
```

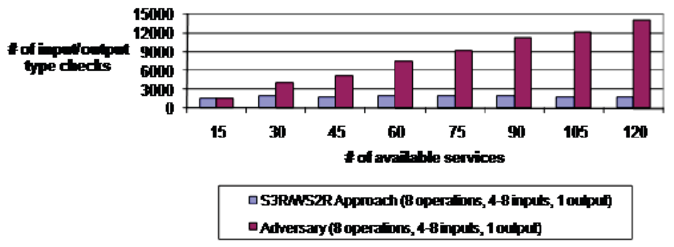
Figure 3. Example adapter for GoogleScholar and Citeseer.

In our example, the adapter of Figure 3 is generated in the case where the `target` service is GoogleScholar and the retrieved `adaptee` is Citeseer; its realization is based on the mappings retrieved according to Theorem 2. 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.

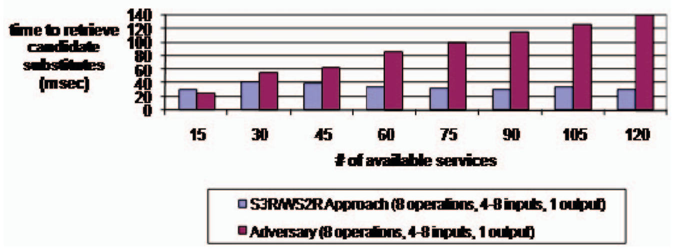
III. Evaluation

To assess the proposed approach we performed two sets of experiments. In both sets we compared the proposed approach, against a typical adversary inspired by service substitution approaches that do not rely on the organization proposed in Section 2. The adversary assumes a registry of available services and tries to retrieve all possible candidate substitutes 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* towards inspecting and validating equivalence and subtyping relations, between user-defined input/output data types. Hence, in this set we measured the number of input/output data type comparisons performed 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*, i.e. the services involved in the substitution scenarios were using input/output data of standard XML types. In both sets of experiments the cardinality of available services in the registries varied in the range [15, 120]. 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 4(a) (1st set) gives the the mean (over the 500 target services used) number of input/output type checks performed to retrieve candidate substitute services. In the case of the proposed approach, the 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, in Figure 4(b) (2nd set) we can observe



(a) 1st set.



(b) 2nd set.

Figure 4. Experimental results.

that the mean 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.

IV. Conclusion

In this paper, we proposed a framework that reduces the complexity of the service substitution problem. Our experimental results highlighted the aforementioned benefit. Currently, 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 [6]. The proposed framework may be extended to account for mismatches in the order of operations; it may also be combined with keywords-based and QoS-based search techniques.

Acknowledgment

This work is done within the context of the ForeverSOA team as part of by the "Equipes Associees" Program of INRIA. The work is partially supported by the CONNECT project (www.connect-forever.eu), which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the ICT theme of the Seventh Framework Programme for Research of the European Commission, under grant agreement number 231167. Finally, we acknowledge the support of the UoI Research Committee.

References

[1] 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.

[2] 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.

[3] L. Baresi, C. Ghezzi, and S. Guinea, *Studies in Computational Intellligence*. Springer, 2006, vol. 42, ch. Towards Self-healing Compositions of Services.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] M. Colombo, E. D. Nitto, and M. Mauri, "SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules," in *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC'06)*, 2006.

[10] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

[11] 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.

[12] A. Zisman, G. Spanoudakis, and J. Dooley, "A Framework for Dynamic Service Discovery," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2008.

[13] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL," in *Proceedings of the 17th International World Wide Web Conference (WWW'08)*, 2008.

[14] B. Liskov and J. Wing, "A Behavioral Notion of Subtyping," *ACM Transactions on Programming Languages and Systems (ACM TOPLAS)*, vol. 16, no. 6, pp. 1811–1841, 1994.

[15] 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.