HAL
open science

Hiding Design-Decisions in Service-Oriented Software via Service Abstraction Recovery

Dionysis Athanasopoulos, Apostolos Zarras, Valérie Issarny, Panos Vassiliadis

To cite this version:

Dionysis Athanasopoulos, Apostolos Zarras, Valérie Issarny, Panos Vassiliadis. Hiding Design-Decisions in Service-Oriented Software via Service Abstraction Recovery. [Research Report] 2010. inria-00491349v2

HAL Id: inria-00491349

https://inria.hal.science/inria-00491349v2

Submitted on 14 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Hiding Design-Decisions in Service-Oriented Software via Service Abstraction Recovery

Dionysis Athanasopoulos
Dep. of Computer Science - Univ. of Ioannina Greece
INRIA-Paris-Rocquencourt
dathanas@cs.uoi.gr

Apostolos V. Zarras
Dep. of Computer Science - Univ. of Ioannina Greece
zarras@cs.uoi.gr

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

Panos Vassiliadis
Dep. of Computer Science - Univ. of Ioannina Greece
pvassil@cs.uoi.gr

ABSTRACT

In this paper, we propose an approach for the recovery of service abstractions out of sets of available services that play the role of alternative design-decisions, which can be used in a service-oriented application. A service abstraction provides a uniform interface that hides differences in the interfaces of alternative services and consequently allows reducing the coupling between the application and the services. To this end, we formally define the notion of service abstraction and propose a hierarchical clustering algorithm that incrementally recovers a hierarchy of service abstractions out of a given set of alternative design-decisions/services. Finally, we evaluate the proposed algorithm with real-world sets of services and report on our findings.

1. INTRODUCTION

The Service-Oriented Architecture (SOA) paradigm emerged as a promising solution to the rapid, low-cost development of software applications that integrate independent, reusable functionalities, designed, implemented and maintained by third parties [17]. According to SOA, existing functionalities are exposed as services that provide programmable interfaces. Then, the design of an application consists of a composition of such services, which is implemented via invocations on the services' operations. From a technical perspective, the Web services technology allows offering services that are accessible through the Web infrastructure [18].

SOA promotes the availability of alternative design-decisions: The synergy of these conceptual and technological advances results in a large amount of services that are published in public service registries (e.g., ServiceFinder[1], WebServicesList[2], RemoteMethods[3]). The contents of these registries are divided into different categories that include information about alternative design-decisions/services, which offer the same/similar functionality, through different programmable interfaces. Hence, application developers may conceive service compositions that benefit from the conjunction/disjunction of alternative services. This is useful in various cases; a service composition that consists of the conjunction of alternative services may serve for collecting and comparing data, offered by these alternative services, while a service composition that consists of the disjunction of alternative services may serve for handling service failures or service quality deterioration.

In general, we can distinguish two different categories of services that offer similar functionalities through different programmable interfaces. In the first category, we have alternative versions of a service that come from the same service provider [15]. In the second category, we have services developed by different service providers [14]. Taking a concrete example, in the RemoteMethods registry we may find various similar services that allow sending SMS messages to mobile phones. `SMS-TXT`[4] is a service, for which there exist different versions from the same service provider. These versions differ in both the communication protocols and the interfaces used for accessing the service functionality. The interfaces of two `SMS-TXT` versions are given in Figure 1(a). Both versions provide a single operation named `SendSms`. Nevertheless, in the `SendSmsHttpPost` interface the operation accepts as input 5 string parameters, while in the `SendSmsSoap` interface the same input data are organized in a single parameter that corresponds to a complex data type. On the other hand, `GlobalSMSPro`[5] is a service that comes from a different service provider and offers a more complex interface, named `SMSTextMessagingSoap` (Figure 1(a)). This interface consists of 6 operations; `SendMessage` and `SendMessagesBulk` are the operations that actually send SMS messages to mobile phones, while the rest of the operations serve for monitoring the status of SMS messages, or finding information concerning different country codes, network standards, etc.

[1] http://demo.service-finder.eu/index

[2] http://www.webservicelist.com/

[3] http://www.remotemethods.com/

[4] http://www.sms-txt.co.uk/sendSms.asmx

[5] http://www.strikeiron.com/Apps/runapp.aspx?appid=95

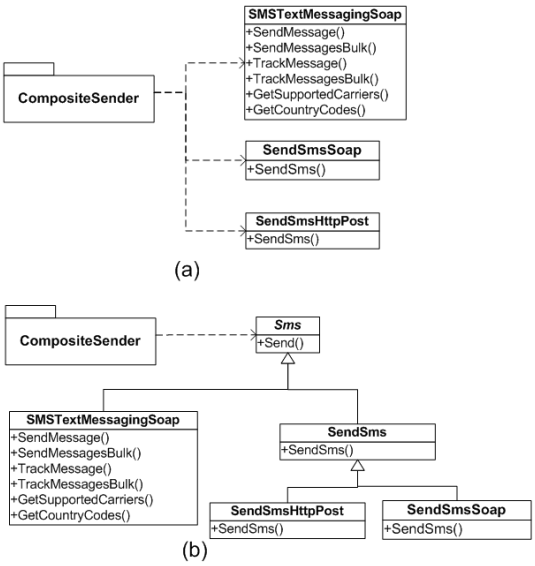

Figure 1: Motivating example.

Using alternative design-decisions increases the application to services coupling: From the application developers' perspective, the differences between the interfaces of alternative services should be handled so as to integrate these services in a particular application. Technically, the developer must implement a piece of code for every alternative service, at each point where the application needs to invoke an operation offered by this service (e.g., the `SendSms` operation of `SMS-TXT` and the `SendMessage` operation of `GlobalSMSPro`). Moreover, to incorporate in the application alternative services that will be found in the future, the corresponding points in the application must be modified accordingly. This naive approach implies that the application will be coupled with the interface of every alternative design-decision/service. In our example, for instance, the `CompositeSender` application is coupled with all three alternative options (Figure 1(a)).

Coupling can be reduced via the fundamental concept of abstraction: Of course the high degree of coupling between the application and the interfaces of alternative services can be avoided, by following a more sophisticated approach inspired by the fundamental principle of information hiding [13]; the typical technique to reduce coupling between an application and a set of alternative design-decisions that solve the same problem differently is to hide these design-decisions behind an abstraction. Using this concept in SOA amounts to hiding the differences of the interfaces of alternative services behind service abstractions. In simple words, a service abstraction should provide a unified interface that represents common/similar functionalities, offered by the different interfaces of the various alternative services. The service abstraction should be accompanied by a corresponding implementation of the unified interface that maps invocations on the operations of the unified interface, into invocations of corresponding operations, provided by the interfaces of the represented services. To complete the picture, the application developer must develop the application with respect to the service abstraction interface, instead of building it with respect to the different interfaces of the alternative services. Having developed the application that way, reduces coupling and consequently allows to easily change, add or remove alternative services, without affecting the application.

The way to define abstractions in SOA is harder than in traditional software development paradigms: The process of defining abstractions in SOA is much different compared to the one followed in traditional software development paradigms (e.g., object-oriented, component-based). Typically, in traditional paradigms we follow a principled, top-down approach, starting from the definition of an abstraction, which is followed by the implementation of the various concrete design-decisions that are hidden behind it. On the contrary, in SOA a bottom-up approach should be followed to take advantage of the notion of abstraction. In SOA, the various design-decisions/services are already there; consequently, abstractions must be reverse-engineered out of the existing available design-decisions. In our example, to reduce coupling between the `CompositeSender` application and the services involved, we have to reverse-engineer a service abstraction (e.g., `Sms` in Figure 1(b)), which offers a unified interface for the common functionality of the services. To this end, a first issue is to identify the operations of the different interfaces that realize similar/same functionality (e.g., the `SendSms` operations of the two versions of `SMS-TXT` and the `SendMessage` operation of `GlobalSMSPro`). Then, the second issue is to define, if possible, a corresponding unified operation (e.g., the `Send` operation in Figure 1(b)) for this common functionality. Concerning the implementation that typically accompanies a service abstraction, different approaches may be assumed. In certain cases it may be useful to develop a single implementation that provides access to all of the alternative services based on certain criteria. In other cases, it may be preferable to have a separate abstraction implementation per different alternative service.

In this paper, we investigate the issue of systematically recovering service abstraction definitions, along with mappings between the abstractions' unified interfaces and the interfaces of the represented alternative services. We do not focus on the generation of service abstraction implementations as this is more or less straightforward given the definition of a service abstraction and the mappings between the unified interface of the abstraction and the interfaces of the represented alternative services.

Contribution: Overall, the systematic definition of service abstractions out of existing services is a complex problem that should be handled to reduce coupling between service-oriented applications and alternative design-decisions/services. Its complexity is the reason for which currently there is no automated solution; the state of the art approaches that employ the concept of service abstractions, rely for their definition on manual procedures, performed by the application developers and/or the service providers [10, 16, 19, 3, 2]. Going beyond the state of the art, in this paper we propose an automated approach for the recovery of service abstractions. To this end, we formally define the notion of service abstractions and propose a hierarchical clustering algorithm. The algorithm accepts as input a set of alternative services, found in a particular category of a public service registry, and incrementally recovers a corresponding hierarchy of service abstractions. Moreover, we evaluate the

proposed algorithm in real-world data sets [7].

The rest of this paper is structured as follows. Section 2 discusses the contribution of the proposed approach with respect to the state of the art. Following, Section 3 formally defines the notion of abstraction in SOA. Section 4, details the abstraction recovery algorithm. Finally, Section 5 provides the results of our experimental evaluation and Section 6 concludes this paper with a summary of our contribution and the future perspectives of this work.

2. RELATED WORK

Abstraction-based development in SOA is a promising solution towards decreasing the coupling between applications and services and facilitating the maintenance [5, 3, 2] and the validation [4] of service-oriented software. The main idea behind abstraction-based development in SOA is to define higher level abstractions, beyond service interfaces, and develop applications based on these abstractions.

Specifically, in [10] the authors propose a framework that allows defining has-a abstractions, which are called service composition patterns. A composition pattern can be refined into various alternative concrete service compositions. Hence, a client application developed with respect to the composition pattern can exploit any of these alternatives without being directly coupled with them. A similar approach that involves has-a abstractions is proposed in [19].

Moreover, in [16] the authors propose defining is-a abstractions, called abstract services. An abstract service represents a set of alternative services that offer similar functionality, via different interfaces. The abstract service offers an interface that can be mapped into the interfaces of the alternative services. Then, an application, developed with respect to the abstract service may use, via the interface of the abstract service, any of the alternative services, without being directly coupled with them.

Going one step further, in [3] we discuss the need for automating the extraction of service abstractions out of existing services. To this end, in [2] we propose an automated approach for the recovery of service abstractions, which aims at reducing the complexity of the service substitution problem [14, 15, 6, 1, 11]. Nevertheless, in the approach proposed in [2] the notion of service abstraction is very restrictive and the corresponding abstraction recovery technique very primitive, allowing the extraction of abstractions that can only represent services which provide equivalent interfaces. Practically, these are mainly interfaces of alternative versions of the same service.

In this paper, we propose an approach that automates the abstraction recovery process, while relaxing the aforementioned constraints. In particular, the proposed abstraction recovery algorithm belongs to the broader family of hierarchical clustering algorithms that have been widely used for the recovery of software architectures, the identification of objects, the discovery of configuration structures, etc.[9]. In the context of SOA, clustering has been employed to facilitate the discovery of services [7]. However, the goal in this case was to group services based on similar keywords used in their descriptions, as opposed to the discovery of abstractions that offer a unified interface for alternative services.

3. BASIC CONCEPTS

To formally define the notion of service abstraction we rely on a generic conceptual model, which is in line with the W3C standards for SOA [18]. The purpose of the model is to reflect the core concepts of services.

Table 1: Definitions of basic concepts.

$$PortType = (n : string, O) \quad (1)$$
$$O = \{Op_i : Operation | i = 1, \ldots, N\})$$
$$Operation = (n : string, In : Message, Out : Message)$$
$$Message = \{part_i : Part | i = 0, \ldots, M\}$$
$$Part = (n : string, type : BuildinType, lower : int, upper : int)$$

$$Abstraction = (I : PortType, D, M) \quad (2)$$
$$D = \{s_i : PortType | i = 1, \ldots, 2\}$$
$$M = \{m_{s_i} | \forall s_i \in D\}$$
$$m_{s_i} = (mop_{s_i}, M_{io})$$
$$mop_{s_i} : I.O \rightarrow s_i.O$$
$$M_{io} = \{m_{io_k} | \forall op_k \in I.O\}$$
$$m_{io_k} = (mi_{ki}, mo_{ki})$$
$$mi_{ki} : op_k.In \rightarrow mop_{s_i}(op_k).In$$
$$mo_{ki} : op_k.Out \rightarrow mop_{s_i}(op_k).Out$$

Specifically, we assume that a service may expose functionality through a set of interfaces. An *interface* (i.e., a PortType - Table 1), is specified in terms of its name and the set of its *operations*. Each *operation*, is characterized by a name and it is associated with an *input message* and an *output message*. A message is hierarchically structured, consisting of a number of parts, characterized by their names, their XML data types and their upper/lower multiplicity bounds; a message may also be empty. A data type of a particular part could be either built-in or complex (i.e., a hierarchically structured element, consisting of further build-in or complex data types). Notably, for the case of abstraction recovery, the notion of message (Table 1) is defined exclusively by the names and the build-in data types of the leaf elements of the message's hierarchical structure. The reason behind this choice is that the particular structure of input and output data of an operation adds further complexity, while not providing much useful information for the abstraction recovery. In our example, for instance, both versions of the `SMS-TXT` service (Figure 1(a)) provide an operation named `SendSms`. As previously mentioned (Section 1), even in this case, the input data are structured differently; in the `SendSmsHttpPost` interface, we have 5 string parameters, while in the `SendSmsSoap` interface the same input data are organized in a single parameter that corresponds to a complex data type. Hence, the definitions of Table 2(a),(b) ignore the differences in the structure of the input data, reflecting that the significant input parameters are identical in both interfaces.

The main purpose of a service abstraction is to provide a unified interface for a pair of alternative services. In a sense, the service abstraction corresponds to an abstract type, while the represented services correspond to subtypes of this abstract type. In this context, Liskov & Wing defined in [8] a list of fundamental rules (invariants, history, preconditions, post-conditions, contra-variance, and co-variance), which guarantee that a type S is a correct subtype of a type T. To define the notion of service abstraction we considered these fundamental rules. However, some of these criteria cannot be applied in the case of service abstractions.

Table 2: SMS-TXT service interfaces.

PortType		SendSmsHttpPost
operations		
SendSms		
in.m	name	type
	FromName	string
	FromNumber	string
	ToNumber	string
	Message	string
	Locale	string
out.m	Name	type
	0	0

(a)

PortType		SendSmsSoap
operations		
SendSms		
in.m	name	type
	FromName	string
	FromNumber	string
	ToNumber	string
	Message	string
	locale	string
out.m	name	type
	0	0

(b)

Table 3: A part of the `GlobalSMSPro` interface.

PortType		SMSTextMessagingSoap				
operations						
SendMessage				SendMessageBulk		
in.m	name	type		in.m	name	type
	ToNumber	string			ToNumbers	string[*]
	FromNumber	string			FromNumber	string
	FromName	string			FromName	string
	MessageText	string			MessageText	string
out.m	name	type		out.m	name	type
	ToNumber	string			ToNumber	string[*]
	TrackingTag	string			TrackingTag	string[*]
	StatusCode	int			StatusCode	int[*]
	StatusText	string			StatusText	string[*]
	StatusExtra	string			StatusExtra	string[*]

Specifically, the invariants and history rules are state-related constraints, which are not applicable to services since their descriptions do not reveal state information. Concerning the pre-conditions and post-conditions rules [8], certain service description languages provide means for specifying pre-conditions and post-conditions. However, our experience with several collections of available services, shows that pre-conditions and post-conditions are rarely provided. Hence, in the definition of service abstraction we mainly consider the contra-variance and co-variance rules. Briefly the *contra-variance rule* states that every method m_S of the type S is mapped to a method m_T of the type T such that the type of each argument of m_S is a subtype of the type of the corresponding argument of m_T. The *co-variance rule* states that the type of the result of m_T is a subtype of the result of m_S. In the context of SOA, we assume that methods correspond to interface operations, whose only argument is their input message, while their result is their output message (Table 1(1)). Then, we define a service abstraction as follows:

Definition 1. Service Abstraction: A service *abstraction* is defined as a tuple that consists of: a set of interfaces, D, provided by a pair of alternative services; a unified interface I whose operations represent common/similar operations, offered by the interfaces of D; a set of mappings, M, between I and the interfaces of D (Table 1(2)). Specifically, M (Table 1(2)) is defined as a pair of tuples. Each such tuple m_{s_i} corresponds to an alternative service interface $s_i \in D$ and consists of:

- An injection mop_{s_i} (i.e., a one-to-one function) between the operations of I and the operations of s_i.
- A set of mappings M_{io} between the inputs/outputs of the operations of I and the inputs/outputs of the corresponding operations of s_i, which preserve the contra-variance/co-variance rules. In particular, M_{io} (Table 1(2)) is a set of tuples; each such tuple m_{io_k} corresponds to an operation op_k of I and consists of:
  - An injection mi_{ki} that maps an input $i_k \in op_k.In$, to an input $i_{s_i} \in mop_{s_i}(op_k).In$ such that[6]: $(i_k.type \subseteq i_{s_i}.type) \vee (i_k.upper \leq i_{s_i}.upper) \vee (i_k.lower \geq i_{s_i}.lower)$
  - An injection mo_{ki} that maps an output $o_k \in op_k.Out$ to an output $o_{s_i} \in mop_{s_i}(op_k).Out$, such that: $(o_{s_i}.type \subseteq o_k.type) \vee (o_{s_i}.upper \leq o_k.upper) \vee (o_{s_i}.lower \geq o_k.lower)$

[6]Note that we use $t_1 \subseteq t_2$ to denote that either t_1 is equivalent with t_2 (e.g., both types correspond to the XML `string` build-in type), or t_1 is a subtype of t_2 (e.g., t_1= `normalizedString` which is subtype of t_2 = `string`)

Given the previous definition of the notion of service abstraction certain remarks should be underlined. First, we must note that the interface $a.I$ of a particular service abstraction a may provide fewer operations than the interfaces of the alternative services that belong to $a.D$. This is due to the fact that the operations of $a.I$ represent a subset of the common/similar operations of the alternative services. For similar reasons an operation op_k of $a.I$ requires an input message that consists of a unified set of input elements and produces an output message that consists of a unified set of output elements, which are common/similar in the pair of represented operations. The input elements that are not common in the represented operations may also be included in the input message of op_k. However, the default choice of the proposed approach is not to do so, in order to avoid recovering definitions of abstraction interfaces with very large numbers of inputs which may be confusing for application developers. On the other hand, the price to pay for this choice is that an implementation of the unified interface must deal with missing input elements (e.g. by providing default values), when mapping invocations on op_k, into invocations of the represented operations.

Finally, we must note that according to the definition of the notion of abstraction, the interface $a.I$ of a may be included in the set of interfaces $a'.I$ of another service abstraction a'. In other words, it is possible to define a higher level service abstraction to represent pairs of lower level service abstractions, which in turn represent pairs of alternative services. Hence, the notion of abstraction enables the definition of hierarchically structured service abstractions. Practically, using higher-level abstractions in service-oriented applications introduces more benefits concerning the corresponding decrease in the coupling between applications and services, due to the fact that higher-level abstractions represent more services. In particular, the coupling reduction is defined as follows:.

Definition 2. Coupling reduction (CR): The coupling reduction that can be achieved from using a service abstraction a in the design of a particular application, instead of the services that are represented by a, is $CR(a) = N - 1$, where $N \geq 2$ is the number of leaf nodes (i.e., the total number of concrete service interfaces, represented by a) of the hierarchy, whose root is a.

In our example, for instance, (Figure 1(b)) `SendSms` is a lower level abstraction that represents the 2 different `SMS-TXT` interfaces, while `Sms` is a higher level abstraction that represents the 2 different `SMS-TXT` interfaces and the interface of `GlobalSMSPro`. Using the `Sms` abstraction (Figure 1(b)) in the `CompositeSender` application, instead of the 3 service interfaces that are represented by `Sms` results in 1 dependency between the `CompositeSender` and `Sms`, instead of 3 dependencies (Figure 1(a)) between the `CompositeSender` and the represented interfaces. Hence the overall coupling reduction is 2.

4. ABSTRACTION RECOVERY IN SOA

Given a set of alternative design-decisions/services, the goal of the proposed abstraction recovery algorithm is to define service abstractions for alternative services and organize the recovered abstractions hierarchically as previously discussed. Nevertheless, in general there may be different combinations of alternative services that result in multiple candidate definitions of service abstractions (e.g., combining the two versions of `SMS-TXT` *vs.* combining a version of `SMS-TXT` with `GlobalSMSPro`). For that reason the algorithm is an iterative process: during each step the best combination of services is selected for the definition of a corresponding abstraction. Roughly, *the best combination is the pair of services that results in the most representative abstraction, i.e., the abstraction that is most similar to the services that it represents.* Generally, the degree of similarity/relevance of the different parts of a whole determines how *cohesive* is the latter. In our case, to be able to assess different candidate abstractions, we define in Section 4.1 a distance metric, called *lack of cohesion of abstraction (LoCA).* Given this metric, the goal of the overall algorithm is to construct service abstractions by selecting at each step combinations of services that minimize the value of *LoCA*. Following, in Section 4.2, we detail the extraction of candidate service abstractions based on *LoCA*, while in Section 4.3 we discuss the core steps of the algorithm that concern the recovery and hierarchical organization of service abstractions.

4.1 Cohesion of service abstractions

Naturally, *LoCA* assesses the relevance of a particular abstraction to the services that it represents, with respect to both textual (i.e., names of interfaces, operations, input/output elements) and type (types of input/output elements) information. Moreover, *LoCA* is defined, such that its values are in the range $[0, 1]$ with 0 being the value that characterizes abstractions that are as relevant as possible to the services that they represent.

Definition 3. Lack of Cohesion of Abstraction (LoCA): Given a service abstraction a, $LoCA(a)$ is defined as the average of the distances between the interface $a.I$ of a and the interfaces of the alternative services that are represented by a (Table 4(1)). Specifically, the distance $D_I(a.I, s_i)$ between $a.I$ and the interface s_i of an alternative service is defined (Table 4(2)) as the average of the normalized edit distance between the names of the two interfaces[7], and the

[7]Typically, the edit distance between two strings s_1, s_2 with lengths n, m can be defined as $ED(s_1, s_2) = n + m - 2 * lcs(s_1, s_2)$, where $lcs(s_1, s_2)$ is the length of their longest common substring; then, their normalized edit distance is $NED(s_1, s_2) = \frac{2*ED(s_1,s_2)}{n+m+ED(s_1,s_2)}$.

Table 4: Definition of *LoCA*.

$$LoCA(a) = \frac{\sum_{\forall s_i \in a.D} D_I(a.I, s_i)}{|a.D|} \quad (1)$$

$$D_I(a.I, s_i) = \frac{NED(a.I.n, s_i.n)}{2} + \frac{\sum_{\forall op_k \in a.I.O} D_{op}(op_k, mop_{s_i}(op_k))}{2 * |a.I.O|} \quad (2)$$

$$D_{op}(op_k, mop_{s_i}(op_k)) = \frac{NED(op_k.n, mop_{s_i}(op_k).n)}{2} + \frac{D_{io}(op_k, mop_{s_i}(op_k))}{2} \quad (3)$$

$$D_{io}(op_k, mop_{s_i}(op_k)) = \frac{D_i(op_k.In, mop_{s_i}(op_k).In)}{2} + \frac{D_o(op_k.Out, mop_{s_i}(op_k).Out)}{2} \quad (4)$$

$$D_i(op_k.In, mop_{s_i}(op_k).In) = \frac{\sum_{\forall i_k \in op_k.In} D_p(i_k, mi_{ki}(i_k))}{|op_k.In|} \quad (5)$$

$$D_o(op_k.Out, mop_{s_i}(op_k).Out) = \frac{\sum_{\forall o_k \in op_k.Out} D_p(o_k, mo_{ki}(o_k))}{|op_k.Out|}$$

$$D_p(i_k, mi_{ki}(i_k)) = \frac{NED(i_k.n, mi_{ki}(i_k).n)}{2} + \frac{ND_T(i_k.type, mi_{ki}(i_k).type)}{2} \quad (6)$$

$$D_p(o_k, mo_{ki}(o_k)) = \frac{NED(o_k.n, mo_{ki}(o_k).n)}{2} + \frac{ND_T(o_k.type, mo_{ki}(o_k).type)}{2}$$

average of the distances between the operations of $a.I$ and their mappings (i.e., the operations of s_i, determined by the injection mop_{s_i} that belongs to the abstraction mappings $a.M$). In the same way, the distance between an operation $op_k \in a.I.O$ and the mapping of this operation $mop_{s_i}(op_k) \in s_i.O$ is defined as the average of the normalized edit distance between the names of the operations and the average of the distances of their input and output messages (Table 4(3)). Similarly, the distance between the input/output messages of op_k and $mop_{s_i}(op_k)$ is defined as the average of the distances between the constituent elements of the input/output messages of op_k and their mappings, determined by the injections, mi_{ki}, mo_{ki} that belong to the operation input/output mappings M_{io}, (Table 4(4-5)). Finally, the distance between two input/output elements is defined as the average of the normalized edit distance between their names and the normalized distance between their build-in types $type_k, type_{s_i}$ (Table 4(6)). According to Definition 1, these build-in types are related with a subtype relationship (due to the contra-variance or the covariance rules). Consequently, $type_k, type_{s_i}$ are on the same path of the standard XML type hierarchy. Hence, the normalized distance between them is obtained by the absolute subtraction of their depths, divided by the maximum height of the XML type hierarchy, $ND_T(type_k, type_{s_i}) = \frac{|d(type_k)-d(type_{s_i})|}{maxHeight}$.

4.2 Recovering candidate abstractions

Constructing a candidate abstraction ca that represents a given pair of interfaces s_i, s_j amounts to defining the abstraction's interface $ca.I$ and the mapping $ca.M$ between this interface and the represented interfaces. By convention,

the interface $ca.I$ is named with the maximum common substring of the names of the interfaces s_i, s_j.

The complexity of determining the operations that constitute the abstraction interface lies into the fact that the operations of s_i, s_j can be combined into pairs in various ways. Each different pair of s_i, s_j operations corresponds to a different candidate operation that can be part of the abstraction interface. In our example, for instance, the `SendSms` operation of the `SMS-TXT` service (Table 2) can be combined with the `SendMessage` operation of `GlobalSMSPro` (Table 3); alternatively, the `SendSms` operation can be combined with the `SendMessagesBulk` operation of `GlobalSMSPro`. Nevertheless, the overall goal of the algorithm is to construct the abstraction that is most relevant to the services that are represented by this abstraction. In line with this ultimate goal, the algorithm must construct the most relevant interface for the candidate abstraction. In other words, $ca.I$ must be constructed in a way such that the number of operations of $ca.I$ (equiv. the number of pairs of s_i, s_j operations for which the algorithm constructs operations in $ca.I$) is maximized, while $LoCA(ca)$ is minimized. Hence, determining the operations that constitute the abstraction interface amounts to solving an optimization problem and particularly the maximum weighted matching problem in a bipartite graph.

Specifically, let $G_{op} = \{s_i.O \cup s_j.O, CO\}$ be a bipartite graph consisting of two disjointed sets of vertices that represent, respectively, the operations of s_i and the operations of s_j. An edge $op_k \in CO$ that connects a pair of operations $op_{s_i} \in s_i.O, op_{s_j} \in s_j.O$, represents a candidate operation that can be constructed for this pair of operations. Hereafter, we use the term abstract operation to refer to op_k. Moreover, we must note that the bipartite graph may not be complete since it may not be possible to define an abstract operation for every possible pair of op_{s_i}, op_{s_j} due to the contra-variance and the co-variance rules that must hold between op_k, op_{s_i} and op_{s_j}. The edges of G_{op} are weighted; the weight of each edge is the average of the distances between the abstract operation op_k and the operations that are represented by op_k, i.e., $w_{op_k} = \frac{D_{op}(op_k, op_{s_i}) + D_{op}(op_k, op_{s_j})}{2}$.

Having this bipartite graph, the maximum weighted matching problem consists of finding the largest possible set of edges (abstract operations) that do not share common vertices (operations of the s_i, s_j interfaces), such that the sum of the weights of the edges is minimal. Then, according to Theorem 1, the solution to the aforementioned problem is also the solution to the problem of finding the most relevant interface for s_i, s_j.

THEOREM 1. *Let ca be the abstraction defined for the pair of services s_i, s_j such that $ca.D = \{s_i, s_j\}$ and $ca.I.O = O$, where O is the solution of the maximum weighted matching problem for $G_{op} = \{s_i.O \cup s_j.O, CO\}$. Then, $LoCA(ca)$ is minimal with respect to the different combinations of pairs of operations, provided by s_i, s_j.*

PROOF. Given that O is the solution of the maximum weighted matching problem for G_{op} we have that $W_{op} = \sum_{\forall op_k \in O} w_{op_k}$ is minimal with respect to the different combinations of pairs of operations, provided by s_i, s_j. Moreover, given that $ca.I.O = O$ with simple calculations we have: $LoCA(ca) = \frac{NED(ca.I.n, s_i.n) + NED(ca.I.n, s_j.n)}{4} + \frac{W_{op}}{4 * |O|}$. Therefore, $LoCA(ca)$ is also minimal with respect to the different combinations of pairs of operations, provided by s_i, s_j. $\square$

However, in order to construct G_{op} we still have to define candidate abstract operations for the different pairs of operations, provided by s_i, s_j. To do so, each candidate abstract operation op_k that represents op_{s_i}, op_{s_j} is named by the maximum common substring of the names of op_{s_i}, op_{s_j}. Moreover, the input/output elements that constitute the input/output messages of op_k must be defined. Each input/output message element must represent a corresponding pair of input/output message elements of op_{s_i}, op_{s_j}. At this point, we face a similar problem as before. There are many different candidate input/output messages that result from different combinations of the input/output elements of op_{s_i}, op_{s_j}. Consequently, there are different candidate values for the weight w_{op_k} that characterizes op_k in G_{op}.

Obviously, to minimize $LoCA(ca)$ the most relevant input/output message must be defined for op_k, i.e., the value of w_{op_k} should be minimized. In detail, to define the most relevant input/output messages for op_k we have to solve the following optimization problem: for the input and the output messages of op_k the number of elements that constitute them must be maximized, while the w_{op_k} that characterizes op_k in G_{op} must be minimized.

To deal with the aforementioned problem, let $G_{in} = (op_{s_i}.In \cup op_{s_j}.In, C_{In})$ be a bipartite graph consisting of two disjointed sets of vertices that represent, respectively, the input elements of op_{s_i} and the input elements of op_{s_j}. An edge $i_k \in C_{In}$ that connects a pair of input elements $i_{s_i} \in op_{s_i}.In$, $i_{s_j} \in op_{s_j}.In$ represents a candidate input element of $op_k.In$ that can be constructed for this pair of input elements. The element is named by the maximum common substring of the names of i_{s_i}, i_{s_j}. Concerning the type of i_k we have the following cases:

- If the types of i_{s_i}, i_{s_j} are related with a subtype relationship, then the type of i_k is set to the most concrete type of the two (Table 5(1) gives further details for the case where $i_{s_i}.type \subseteq i_{s_j}.type$).
- Otherwise, it is not possible to define an input element that represents i_{s_i}, i_{s_j} in a way that preserves the contra-variance rule, i.e., $G_{in}.C_{In}$ shall contain no edge between i_{s_i}, i_{s_j}. At this point, we must further point out that if the bipartite graph G_{in}, defined for the input messages of op_{s_i}, op_{s_j} contains no edges at all, i.e., $G_{in}.C_{In} = \emptyset$, then we cannot define a candidate abstract operation for op_{s_i}, op_{s_j} i.e., there shall be no edge between these two operations in G_{op}.

The edges of G_{in} are weighted; the weight of each edge is the average of the distances between the input element i_k and the input elements that are represented by i_k, i.e., $w_{ik} = \frac{D_P(i_k, i_{s_i}) + D_P(i_k, i_{s_j})}{2}$. The special case where the input message of op_{s_i} (or op_{s_j}) has no input elements (i.e., the input message is empty) is treated as follows: The input message of op_k is also considered empty; in terms of the graph G_{in} we have that the elements of the input message of op_{s_j} are connected with an empty input element that represents the empty input message of op_{s_i}. The weight of the edges that connect the input elements of op_{s_j}, with the empty input element of op_{s_i} is maximum (i.e., $w_{ik} = 1$).

Similarly, let $G_{out} = (op_{s_i}.Out \cup op_{s_j}.Out, C_{Out})$ be a bipartite graph consisting of two disjointed sets of vertices that represent, respectively, the output elements of op_{s_i} and the output elements of op_{s_j}. An edge $o_k \in C_{Out}$ that connects

Table 5: Definitions of types for the operations of candidate service abstractions.

$$((i_{s_i}.type \subseteq i_{s_j}.type) \vee (i_{s_i}.upper \leq i_{s_j}.upper) \vee (i_{s_i}.lower \geq i_{s_j}.lower)) \Rightarrow ((i_k.type = i_{s_i}.type) \vee (i_k.lower = i_{s_i}.lower) \vee (i_k.upper = i_{s_i}.upper)) \quad (1)$$

$$((o_{s_i}.type \subseteq o_{s_j}.type) \vee (o_{s_i}.upper \leq o_{s_j}.upper) \vee (o_{s_i}.lower \geq o_{s_j}.lower)) \Rightarrow ((o_k.type = o_{s_j}.type) \vee (o_k.lower = o_{s_j}.lower) \vee (o_k.upper = o_{s_j}.upper)) \quad (2)$$

a pair of output elements, $o_{s_i} \in op_{s_i}.Out$, $o_{s_j} \in op_{s_j}.Out$ represents a candidate output element of $op_k.Out$. As in the case of inputs, o_k is named by the maximum common substring of the names of o_{s_i}, o_{s_j}. Regarding the type of o_k we have:

- If the types of o_{s_i}, o_{s_j} are related with a subtype relationship, then the type of o_k is set to the most generic type of the two (Table 5(2) gives further details for the case where $o_{s_i}.type \subseteq o_{s_j}.type$).
- Otherwise, $G_{out}.C_{Out}$ shall not contain an edge between o_{s_i}, o_{s_j} to avoid jeopardizing the co-variance rule. As in the case of inputs, if G_{out}, contains no edges at all, then we cannot define a candidate abstract operation for op_{s_i}, op_{s_j} i.e., there shall be no edge between these two operations in G_{op}.

As in the case of G_{in}, the edges of G_{out} are weighted, i.e., $w_{ok} = \frac{D_P(o_k,o_{s_i})+D_P(o_k,o_{s_j})}{2}$. The special case where the output message of op_{s_i} (or op_{s_j}) is empty is treated similarly to the case of empty input messages.

Given G_{in} and G_{out}, we proceed by solving the maximum weighted matching problem for both of these graphs. In particular, solving the problem in the case of G_{in} amounts to finding the maximum number of edges (input elements for the candidate abstract operation op_k), such that the sum of the weights of the edges is minimal. Solving the same problem for G_{out} consists of finding the maximum number of edges (output elements for the candidate abstract operation op_k), such that the sum of the weights of the edges is minimal. Then, according to Theorem 2, the combination of these two solutions gives the solution to the problem of finding input and output messages for op_k, such that w_{op_k} is minimized.

THEOREM 2. *Let op_k be the abstract operation, defined for the pair of operations $op_{s_i} \in s_i$, $op_{s_j} \in s_j$ and $op_k.In = In$, $op_k.Out = Out$, where, In, Out are the solutions of the maximum weighted matching problem for G_{in}, G_{out}. Then, w_{op_k} is minimal with respect to the different combinations of input/output elements of op_{s_i}, op_{s_j}.*

PROOF. Given that In, Out are, the solutions of the maximum weighted matching problem for G_{in}, G_{out} we have that $W_{in} = \sum_{\forall i_k \in In} w_{ik}$ and $W_{out} = \sum_{\forall o_k \in Out} w_{ok}$ are minimal. In addition, given that $op_k.In = In$, $op_k.Out = Out$ with simple calculations we have:

Table 6: G_{in} graph for the inputs of `SendSms` and the inputs of `SendMessage`.

		SendMessage			
		ToNumber string	**FromNumber string**	**FromName string**	**MessageText string**
SendSms	**FromName string**	o string 0.43	FromN string 0.21	**FromName string 0**	Me string 0.39
	FromNumber string	Number string 0.16	**FromNumber string 0**	FromN string 0.21	m string 0.45
	ToNumber string	**ToNumber string 0**	Number string 0.16	o string 0.43	m string 0.44
	Message string	M string 0.43	M string 0.43	Me string 0.39	**Message string 0.09**
	locale string	o string 0.42	o string 0.43	e string 0.43	e string 0.43

$w_{op_k} = \frac{NED(op_k.n,op_{s_i}.n)+NED(op_k.n,op_{s_j}.n)}{4} + \frac{W_{in}}{8*|In|} + \frac{W_{out}}{8*|Out|}$. Therefore, w_{op_k} is also minimal. □

Overall, the interface of a candidate abstraction ca that represents a given pair of interfaces s_i, s_j is constructed based on the graphs G_{in}, G_{out} and G_{op} that were previously detailed. Technically, the maximum weighted matching problem is solved in all cases based on the fundamental Munkres algorithm [12] that is adapted to the specificities of our problem. To complete the definition of ca, the mappings $ca.M$ are defined based on G_{op} and the graphs G_{in}, G_{out}, defined for every operation op_k of $ca.I.O$. In particular, the injections mop_{s_i}, mop_{s_j} that map the operations of $ca.I$ to the operations of s_i, s_j are derived directly from the sub-graph $G'_{op} = \{s_i.O \cup s_j.O, ca.I.O\}$ of G_{op}; for every pair of nodes op_{s_i}, op_{s_j} that is connected with an edge $op_k \in ca.I.O$ we have that $mop_{s_i}(op_k) = op_{s_i}$ and $mop_{s_j}(op_k) = op_{s_j}$. Similarly, the injections mi_{ki}, mi_{kj} between the inputs of op_k and the inputs of the operations that are represented by op_k are derived from the sub-graph $G'_{in} = (op_{s_i}.In \cup op_{s_j}.In, op_k.In)$ of the graph G_{in}; for every pair of nodes i_{s_i}, i_{s_j} that is connected with an edge $i_k \in op_k.In$ we have that $mi_{ki}(i_k) = i_{s_i}$ and $mi_{kj}(i_k) = i_{s_j}$. Finally, the injections mo_{ki}, mo_{kj} between the outputs of op_k, and the outputs of op_{s_i}, op_{s_j}, are derived directly from the sub-graph $G'_{out} = (op_{s_i}.Out \cup op_{s_j}.Out, op_k.Out)$ of G_{out}.

Returning to our example, Table 6 gives the adjacency matrix that represents the G_{in} graph for the input elements of `SendSms` (Table 2) and `SendMessage` (Table 3). The dark grey cells are the nodes of the graph, while the white cells are the edges. Hence, each white cell corresponds to a candidate input that can be constructed, characterized by its name, type and weight. Solving the maximum weighted matching problem in this graph results in the subset of the most relevant candidate inputs (edges), represented by the light grey cells, which minimize the value of W_{in}.

Table 7, gives the adjacency matrix that represents the G_{op} graph for the `SendSmsSoap` interface of `SMS-TXT` and the `SMSTextMessagingSOAP` interface of `GlobalSMSPro`. The edges correspond to the candidate abstract operations that

Table 7: G_{op} graph for the `SMS-TXT` and the `GlobalSMSPro` services.

	SendSmsSoap	
		SendSms
SMSTextMessagingSoap	**SendMessage**	**Send, 0.53**
	SendMessageBulk	Send, 0.57
	TrackMessage	S, 0.78
	TrackMessageBulk	S, 0.79
	GetCountryCodes	S, 0.94
	GetSupportedCarriers	S, 0.95

Table 8: Examples of candidate service abstractions.

PortType	SendSms	
operations		
SendSms		
in.m	**name**	**type**
	FromName	string
	FromNumber	string
	ToNumber	string
	Message	string
	locale	string
out.m	**name**	**type**
	0	0

(a)

PortType	Sms	
operations		
Send		
in.m	**name**	**type**
	FromName	string
	FromNumber	string
	ToNumber	string
	Message	string
out.m	**name**	**type**
	0	0

(b)

can be constructed, when we combine these two interfaces towards constructing a corresponding candidate service abstraction. Solving the maximum weighted matching problem in this graph results in a single edge that corresponds to an operation named `Send` which represents the `SendSms` and the `SendMessage` operations. The detailed description of the overall candidate abstraction for the aforementioned two services is given in Table 8(b).

4.3 Core steps of the algorithm

The abstraction recovery algorithm accepts as input a set of service interfaces $S = \{s_i : PortType | i = 1, \ldots, K\}$, provided by alternative services. The output of the algorithm is a set of hierarchically structured service abstractions $A = \{a_l : Abstraction | l = 1, \ldots, L\}$. To this end, the algorithm iteratively performs the following steps:

1. For every pair of interfaces $s_i \in S, s_j \in S$ the algorithm attempts to extract the most relevant candidate abstraction ca that conforms to Definition 1. This may not be possible for every pair of service interfaces due to the contra-variance and the co-variance rules that must hold between the interface of the abstraction and the interfaces of the represented services. Hence, the outcome of this step is a set of candidate abstractions CA. For every candidate abstraction $ca \in CA$, $LoCA(ca)$, is calculated.

2. Among all the extracted candidate service abstractions the algorithm selects $ca \in CA$ that is characterized by the minimum lack of abstraction cohesion, i.e., $\forall ca' \in CA | ca' \neq ca \Rightarrow LoCA(ca') \geq LoCA(ca)$.

3. The selected abstraction ca is included in the result, i.e., $A = A \cup \{ca\}$. Moreover, the services that are represented by ca are removed from the input set, i.e., $S = S - ca.D$. Finally, $ca.I$ is included in S, i.e., $S = S \cup \{ca.I\}$, so as to serve for the recovery of higher level abstractions.

4. The algorithm repeats steps (1) to (4), until the input set comprises only one element, namely, the root abstraction of the resulting abstraction hierarchy A, which generalizes all the available service interfaces, or until no further abstractions can be recovered.

Given that the proposed algorithm is a variant of agglomerative clustering, its complexity is $O(|S|^2 * \log(|S|))$.

Getting back into our example, suppose that the input to the proposed algorithm comprises the interfaces of the two versions of the `SMS-TXT` service (Table 2) and the interface of the `GlobalSMSPro` service (Table 3). During its first step, the algorithm constructs a candidate abstraction ca_1 that represents the interfaces of the two versions of the `SMS-TXT` service (Table 8(a)). The interface of ca_1 comprises a single operation that represents the two `SendSms` operations of the represented interfaces. The value of $LoCA$ for ca_1 is 0.23. Two more candidate abstractions ca_2, ca_3 are constructed based on the interface of `GlobalSMSPro` and the interfaces of the two versions of the `SMS-TXT` service. The interfaces of ca_2, ca_3 are identical (Table 8(b)) and the values of $LoCA$ for them are 0.62, 0.68.

Therefore, according to $LoCA$, ca_1 is the most representative abstraction, which is the outcome of the first iteration of the algorithm. This is a reasonable result which reflects the fact that ca_1 represents the interfaces of two different versions of the same service, while ca_2, ca_3 represent pairs of independently developed services. During the second iteration, ca_1 can be further combined with the `GlobalSMSPro` service to produce a higher level abstraction that represents the interfaces of all the alternative services. Overall, Figure 1(b) gives the results of the abstraction recovery process for the services of our example.

5. EVALUATION

To assess our abstraction recovery approach we used the proposed algorithm to reverse engineer service abstractions out of the woogle data set [7], which comprises a rich variety of services that have been retrieved using the woogle Web crawler. The goal of the assessment was to investigate the quality of the results produced by the proposed approach, which was evaluated with respect to: (1) the well-formedness of the interfaces of the extracted abstractions and their mappings to the interfaces of the represented services; (2) the coupling reduction CR (Definition 2) that can be achieved from the abstractions; (3) the $LoCA$ (Definition 3) that characterizes the abstractions. An abstraction is considered *well-formed* if the operations of the abstraction interface are mapped into corresponding operations of the represented services that constitute alternative options for the same functionality.

In our experiments we used 6 different sets of alternative design-decisions/services. Each one of the 6 sets included the contents of a different category of woogle services. A brief description of each input set is given in Table 9. Cer-

Table 9: Input sets descriptions.

Category	# service interfaces	Description
SMS	14	sending SMSs to mobile phones
Email	16	calendar services
WHOIS	14	find info for people
Bank	8	Check credit card information
Content	41	weather, news services
Utilities	25	math, search engines, etc.

Table 10: Experimental results: ranges of *LoCA* & *CR* per input set.

Abstractions Categories \ Service Categories		*SMS*	*Email*	*Bank*	*WHOIS*	*Content*	*Utilities*
Same provider – well formed		0.05-0.23	0.06-0.33	0.09-0.16	0.12-0.23	0.08-0.21	0.06-0.17
Different provider	*well formed*	0.33-0.46	0.36-0.65	0.30	0.03-0.34	0.0-0.33	0.28-0.54
	not well formed	-	0.5	0.71	0.73-0.85	0.50-0.875	0.50-0.66

(a) Ranges of $LoCA$.

Abstractions Categories \ Service Categories		*SMS*	*Email*	*Bank*	*WHOIS*	*Content*	*Utilities*
Same provider – well formed		1-2	1-2	1-2	1-2	1-2	1-2
Different provider	*well formed*	5-13	5	4	1-9	1-5	1-3
	not well formed	-	6	7	3-13	3-40	1-16

(b) Ranges of CR.

tain categories (i.e., Content and Utilities) were more noisy than the others, in the sense that they were quite broad, containing services that were not much relevant with each other.

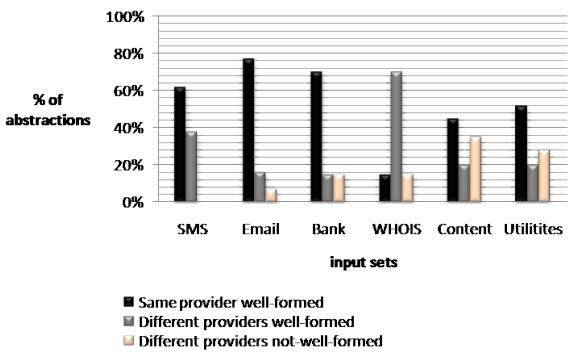

Figure 2: Experimental results: well-formed vs. non well-formed abstractions.

Quality of results: Figure 2 and Table 10 provide a summary of the results that we obtained from the 6 input sets of services that were used in our experiments. Specifically, Figure 2 compares the percentages of well-formed and non-well-formed abstractions. Table 10(a) gives the ranges of the values of $LoCA$ for the recovered abstractions per different input set. Table 10(b) shows the ranges of CR for the recovered abstractions per different input set. Finally, Figure 3 provides a more detailed view of the abstraction hierarchy that was extracted for the case of the SMS input set.

Concerning the quality of the results, we performed a detailed review of the abstractions that were extracted from each input set. The main observation of the review was that the extracted abstractions in all input sets can be divided in 3 main categories. The first category consists of abstractions characterized by relatively small values of $LoCA$ (< 0.2 in all cases) and CR (< 3 in all cases). Their percentages range for the different input sets from 15% to 77%. All these abstractions are well-formed and represent interfaces of different versions of the same service. The second category comprises abstractions characterized by higher values of $LoCA$ (< 0.65 in all cases) and CR (< 9 in all cases). Their percentages range for the different input sets from 15% to 70%. These abstractions are also well-formed. The interfaces of the represented services are offered by different service providers. Finally, the third category consists of non-well-formed abstractions, in the sense that certain operations of the abstractions interfaces were mapped into irrelevant operations of the represented services. The percentages of these abstractions range per different input set from 0% to 35%. These percentages are quite reasonable, considering that the results were produced by a fully automated process. As expected the highest percentages of non-well-formed abstractions were observed in the most noisy input sets (i.e., Content and Utilities), while in most other cases the percentages of non-well-formed abstractions were less than 15%. The values of $LoCA$ for all the non-well-formed abstractions were greater than > 0.5.

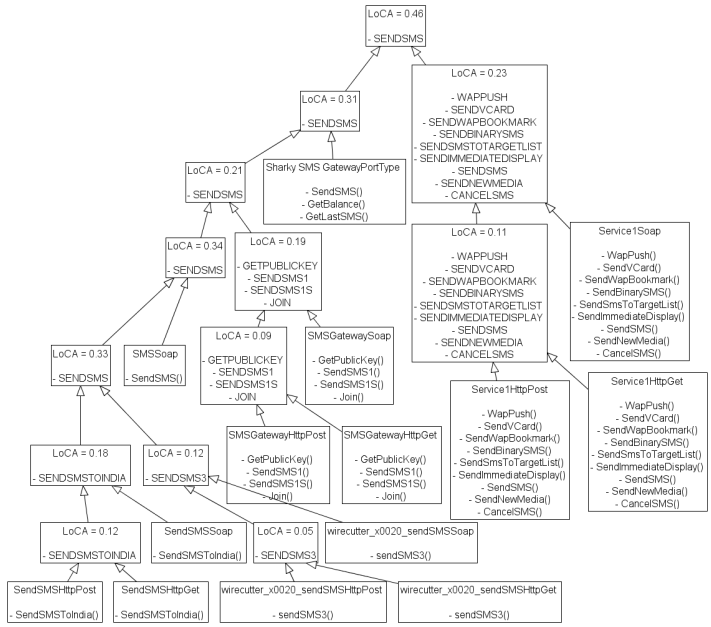

Figure 3: Recovered abstraction hierarchy for the SMS input set.

Therefore, we can conclude that the proposed approach produced for all input sets a high percentage of well-formed abstractions (ranging from 65% to 100% for the 6 input sets) that reduce the coupling between applications and services. A smaller percentage of these abstractions (ranging from 15% to 70% for the 6 input sets) is even more beneficial

since these abstractions are characterized by higher values of CR. Concerning $LoCA$, we can conclude that there is no threshold that distinguishes for sure well-formed from non-well-formed abstractions. However, our experience with the abstractions recovered from the 6 input sets indicates that abstractions with $LoCA < 0.3$ are usually well-formed, while abstractions with $LoCA > 0.7$ are good candidates to be filtered out from the results produced by the proposed algorithm.

Threats to validity: Concerning our experimental conclusions, an obvious threat to internal validity is the definition of $LoCA$ that we assume for the recovery of abstractions. A different definition (e.g., a weighted scheme of name and type distances) may affect the extracted abstractions. Nevertheless, the results we obtained with the current definition of $LoCA$ are already very encouraging. A threat to external validity is the input sets of our experiments. To alleviate this threat we used input sets from a real-world set of services that have been collected by crawling the Web and used in a previous widely accepted research work [7].

6. CONCLUSION

In this paper, we proposed an approach for the recovery of service abstractions out of sets of alternative design-decisions/services. The recovered abstractions provide a uniform interface that hides differences in the interfaces of the services that are represented by these abstractions and consequently allow reducing the coupling between application and services. We evaluated the proposed approach with real-world sets of services. Our results showed that a high percentage of the recovered abstractions are actually useful. Nevertheless, the proposed approach also produces a percentage of useless abstractions that should be filtered out by the developers by inspecting the abstractions' inherent characteristics.

Given the current status of this work, we currently work towards a systematic approach for reducing the number of useless abstractions. Moreover, our future research plans include a late-binding mechanism that would actually enable polymorphism based on recovered service abstractions and the services that are represented by these abstractions. Finally, we plan to extend the notion of service abstraction to further reflect quality aspects of the represented services and account for such aspects in the proposed abstraction recovery approach.

7. REFERENCES

[1] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A Framework for Executing Adaptive Web-Service Processes. *IEEE Software*, 24(6):39–46, 2007.

[2] D. Athanasopoulos, A. Zarras, and V. Issarny. Service Substitution Revisited. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2009. (short paper).

[3] 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)*, 2009.

[4] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA testing. In *Proceedings of the 7th European Software Engineering Conference / ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 161–170, 2009.

[5] D. Bianculli, R. Jurca, W. Binder, C. Ghezzi, and B. Faltings. Automated Dynamic Maintenance of Composite Services Based on Service Reputation. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC)*, pages 449–455, 2007.

[6] 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)*, pages 191–202, 2006.

[7] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *Proceedings of VLDB*, 2004.

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

[9] O. Maqbool and H. Babri. Hierarchical Clustering for Software Architecture Recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780, 2007.

[10] L. Melloul and A. Fox. Reusable Functional Composition Patterns for Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 498–506, 2004.

[11] 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)*, pages 815–824, 2008.

[12] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[13] D. Parnas. On the Criteria for Decomposing To Be Used for Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[14] S. R. Ponnekanti. Application-Service Interoperation Without Standardized Service Interfaces. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 30–37, 2003.

[15] S. R. Ponnekanti and A. Fox. Interoperability Among Independently Evolving Web Services. In *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE)*, pages 331–351, 2004.

[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)*, pages 166–173, 2006.

[17] E. Thomas. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[18] W3C. Web Services Architecture. Technical report, W3C. http://www.w3c.org/TR/ws-arch.

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