Mining Service Abstractions (NIER Track)

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

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

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

ABSTRACT

Several lines of research rely on the concept of service abstractions to enable the organization, the composition and the adaptation of services. However, what is still missing, is a systematic approach for extracting service abstractions out of the vast amount of services that are available all over the Web. To deal with this issue, we propose an approach for mining service abstractions, based on an agglomerative clustering algorithm. Our experimental findings suggest that the approach is promising and can serve as a basis for future research.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies, representation*

General Terms

Design

Keywords

Abstraction recovery, agglomerative clustering, services

1. INTRODUCTION

Background & State of the Art: In the early 70's, Parnas [12] introduced the fundamental principle of information hiding and discussed the benefits from developing software with respect to abstractions that hide the details of various alternative design options.

Today, all over the Web we have a plentitude of alternative design options, provided in the form of reusable services. The amount of these options is constantly growing. Given the results reported in [1] the annual growth rate of the number of services that become available through the Web is 130%. Today, we further have crawlers and Web service search engines [7, 1] that allow the discovery of large amounts of available design options.

In addition to all these, several lines of research rely on the concept of service abstractions that represent semantically compatible services (i.e., services which provide the same functionality, possibly, through different interfaces). In particular, various semantic description languages have been proposed for the specification of service abstractions (e.g., OWL-S[1], WSMO[2]). Moreover, many service registries (e.g., [8, 13]) assume certain notions of abstraction, towards the organization of information concerning available services. Finally, a large variety of approaches for service composition assume the existence of service abstractions (e.g., [14, 15, 6, 2, 4, 5]). Then, the proposed approaches provide means for adapting compositions of service abstractions to meet changes in functional and non-functional requirements. The adaptation takes place, by substituting the concrete services that are hidden behind the composed service abstractions.

Contribution: Although, several lines of research rely on the concept of service abstractions to enable the organization, the composition and the adaptation of services, what is still missing at this point is a systematic approach for extracting service abstractions out of the vast amount of services that are available all over the Web. To cover this lack, in this paper we propose an approach for mining service abstractions. The core idea is to mine a hierarchy of service abstractions that represent alternative design options, via an agglomerative clustering algorithm that takes as input Web service descriptions gathered by crawling the Web.

Impact: In general, we view the proposed approach as a stand-alone mechanism that may serve as a baseline for the various lines of research that assume the existence of service abstractions towards the organization, the composition and the adaptation of services.

In the rest of the paper, we provide details regarding the proposed approach (Section 2), we discuss our preliminary results (Section 3) and finally we conclude with the future directions of this work (Section 4).

2. MINING SERVICE ABSTRACTIONS

The overall process that mines service abstractions consists of two phases. The first phase accepts as input a set

[1]http://www.w3.org/Submission/OWL-S/
[2]http://www.wsmo.org/

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

$$PortType = (n : string, O) \quad (1)$$
$$O = \{Op_i : Operation\}$$
$$Operation = (n : string, In : Message, Out : Message)$$
$$Message = \{part_i : Part\}$$
$$Part = (n : string, type : BuildinType, lower : int, upper : int)$$
$$Abstraction = (I : PortType, D, M) \quad (2)$$
$$D = \{s_i : PortType\}$$
$$M = \{m_{s_i} : I.O \rightarrow s_i.O\}$$

Figure 1: Definitions of basic concepts.

of Web service descriptions and performs an initial preprocessing that aims at dividing the Web service descriptions in coarse-grained categories of relevant services (e.g., Email services, SMS services, etc.). During the preprocessing phase, the categorization relies on an existing keyword-based clustering algorithm [7]. Then, during the second phase each coarse-grained category of service descriptions is given as input to the proposed agglomerative clustering algorithm that produces as output a hierarchy of service abstractions. Following, we provide basic definitions concerning the concept of service abstractions and discuss the *modus operandi* of the proposed algorithm.

2.1 Basic Concepts

We assume that a service may provide a set of interfaces. An interface (i.e., a *PortType* - Figure 1(1)), is specified in terms of a name and a set of operations, *O*. Each operation, is characterized by a name, an input message, *In*, and an output message, *Out*. In general, 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. The 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). In the proposed mining process we consider only the leaf elements of the message's hierarchical structure. The reason for this choice is that the particular structure of the input and output data of an operation adds further complexity, while not providing much useful information to the mining process. Indeed, in our technical report [3] we give examples of services that offer semantically compatible functionalities, while the structures of their input and output messages are very different.

Ideally, a service abstraction should represent a set of available services that have in common a certain set of semantically compatible functionalities, realized by corresponding sets of operations, which most possibly would be syntactically different. Finding within a given set of services that were gathered by crawling the Web, services that provide common semantically compatible functionalities is very hard. However, we empirically observed that it is very frequently encountered to have semantically compatible services that provide syntactically similar interfaces. Our preliminary experiments in Section 3 provide evidence for this correlation.

Then, to assess the similarity between two service interfaces we rely on a distance metric D_I, which is defined as follows (Figure 2). Given two interfaces s_i, s_j and a mapping

$$D_I(s_i, s_j) = \frac{NED(s_i.n, s_j.n)}{2} + \frac{\sum_{\forall (op_i, op_j) \in M_{op_{ij}}} D_{op}(op_i, op_j)}{2 * |M_{op_{ij}}|} \quad (1)$$

$$D_{op}(op_i, op_j) = \frac{NED(op_i.n, op_j.n)}{2} + \frac{D_{io}(op_i, op_j)}{2} \quad (2)$$

$$D_{io}(op_i, op_j) = \frac{D_m(op_i.In, op_j.In)}{2} + \frac{D_m(op_i.Out, op_j.Out)}{2} \quad (3)$$

$$D_m(m_i, m_j) = \frac{\sum_{\forall (p_i, p_j) \in M_{m_{ij}}} D_p(p_i, p_j)}{|M_{m_{ij}}|} \quad (4)$$

$$D_p(p_i, p_j) = \frac{NED(p_i.n, p_j.n)}{2} + \frac{ND_T(p_i.type, p_j.type)}{2} \quad (5)$$

Figure 2: Distance between service interfaces.

$M_{op_{ij}} \subset s_i.O \times s_j.O$ between the most similar operations of the interfaces, the distance $D_I(s_i, s_j)$ is defined as the average of the normalized edit distance between the names of the interfaces[3], and the average of the distances between the mapped operations. The distance $D_{op}(op_i, op_j)$ between two operations op_i, op_j 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. Given a mapping $M_{m_{ij}} \subset m_i \times m_j$ between the most similar parts of two messages m_i, m_j, we define the distance between the messages as the average of the distances between the mapped parts. Finally, the distance between two message parts is defined as the average of the normalized edit distance between their names and the normalized distance between their build-in types $ND_T(type_i, type_j)$; if these types are in the same branch of the standard XML type hierarchy, then $ND_T(type_i, type_j)$ is the absolute difference of their depths, divided by the maximum height of the XML type hierarchy, otherwise we assume that the types are incompatible and $ND_T(type_i, type_j) = \infty$.

Based on the previous concepts, we define a service abstraction as a tuple that consists of: an abstract interface I and a set of represented service interfaces D (Figure 1(2)). Each operation of the abstract interface I is mapped, through a set of mappings M, to a set of operations, provided by the represented interfaces. Specifically, for each service interface s_i of D, M comprises a one-to-one function m_{s_i} between the operations of I and the operations of s_i. By construction, each mapping m_{s_i} is such that the fundamental contravariance/covariance rules hold for the inputs/outputs of the mapped operations [9].

Finally, it should be noted that, in general, the interface $a.I$ of a service abstraction 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 hierarchy that consists of higher level service abstractions, which represent lower level service abstractions.

[3]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 $NED(s_1, s_2) = \frac{2*ED(s_1,s_2)}{n+m+ED(s_1,s_2)}$.

2.2 Agglomerative Clustering

The ultimate goal of the mining process is to construct the interfaces of service abstractions, along with mappings between these interfaces and the interfaces of the represented services. Consequently, the typical agglomerative clustering algorithms (e.g., SLA, CLA, WLA, ULA [10]) are not directly applicable in our case. Following, we discuss the core steps of the proposed algorithm, while the interested reader may refer to our technical report [3] for further technical details.

The mining algorithm accepts as input a set of interfaces $S = \{s_i : PortType\}$, provided by services that belong in a category of services that resulted from the preprocessing phase. The output of the algorithm is a set of hierarchically structured service abstractions $A = \{a_l : Abstraction\}$. To this end, the algorithm iteratively performs the following steps:

Step 1: For every pair of interfaces $s_i \in S, s_j \in S$ the algorithm finds the distance $D_I(s_i, s_j)$. To this end, the most similar pairs of operations $(op_i, op_j) \in s_i.O \times s_j.O$ (i.e., the mapping $M_{op_{ij}}$ - Section 2.1) are found by solving the maximum weighted matching problem in a bipartite graph [11]. The nodes of the graph correspond to the operations of s_i and s_j, while the edges correspond to the distances between the operations. Finding the distances between two operations $op_i \in s_i.O$ and $op_j \in s_j.O$ involves finding the most similar pairs of elements for the input messages (respectively the output messages) of the operations (i.e., the mapping $M_{m_{ij}}$ - Section 2.1). This problem is also solved by solving the maximum weighted matching problem in a bipartite graph that represents the input messages (respectively the output messages). Note that in this step it is possible to calculate a distance between two interfaces that equals to ∞. This case may come up if the best possible matching between messages results in at least one pair of incompatible types. In such a case, we consider that it is not possible to create an abstraction out of the two interfaces.

Step 2: Based on the calculated distances the most similar pair of interfaces (s_i, s_j) is selected and an abstraction a is constructed as follows: By convention, the name of $a.I$ is the longest common substring of the names of s_i, s_j. For every pair of matched operations (op_i, op_j) found in the previous step, $a.I$ comprises a corresponding operation op_a, named by following the same convention. The input (respectively output) message of op_a, contains a message part p_a for every matched pair (p_i, p_j) of elements of the input (respectively output) messages of op_i, op_j. Concerning the type of the input (respectively, output) element p_a, we have $p_a.type = p_i.type$ if $p_i.type$ is higher (respectively lower) than $p_j.type$ in the standard XML type hierarchy; otherwise, $p_a.type = p_j.type$.

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

Step 4: The algorithm repeats steps (1) to (3), 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.

Table 1: Example.

SendSms	SendMessage			
	ToNumber string	FromNumber string	FromName string	MessageText string
FromName string	0.43	0.21	**0**	0.39
FromNumber string	0.16	**0**	0.21	0.45
ToNumber string	**0**	0.16	0.43	0.44
Message string	0.43	0.43	0.39	**0.09**
locale string	0.42	0.43	0.43	0.43

(a) inputs mapping

SMSTextMessaging	SendSms
	SendSms
SendMessage	**0.53**
SendMessageBulk	0.57
TrackMessage	0.78
TrackMessageBulk	0.79
GetCountryCodes	0.94
GetSupportedCarriers	0.95

(b) Operations mapping

Table 2: Input sets descriptions.

Category	#services/#interfaces	Description
SMS	6/14	sending SMSs
Email	8/16	calendar services
WHOIS	6/14	find info for people
Content	26/41	weather, news services
Utilities	19/25	math, search engines, etc.

Taking a real-world example, assume that the input of the algorithm S contains a simple service, `SMS-TXT`[4], that provides a single operation, `SendSms`, for sending SMS messages. Moreover, S contains a more complex service, `GlobalSMSPro`[5], that provides an operation `SendMessage` for sending SMS messages, along with further operations that serve for various other purposes. Then, Table 1(a) highlights in grey the best possible mapping between the most similar pairs of input elements for the aforementioned two operations. The distances between all the possible pairs of input elements are given in the corresponding cells. Moreover, Table 1(b) highlights in grey the most similar pair of operations for the interfaces of the two services; in particular, `SendSms` is matched with `SendMessage`. Based on this matching, the interface I of the resulted abstraction would comprise a single operation, named `Send` and a mapping M between `Send` and the matched operations.

3. EMERGING RESULTS

To assess the proposed approach we used the mining algorithm to extract service abstractions out of the woogle data set [7][6]. The goal of the evaluation was to investigate the capability of the algorithm in finding useful service abstractions, where by the term useful we mean abstractions that actually represent semantically compatible services. In our experiments, we used as input to the algorithm 5 different categories of services resulted from the preprocessing stage. A brief description of each category is given in Table 2. For each category, we manually inspected the abstractions hierarchies produced by the algorithm and measured the percentages of useful and useless abstractions. The results are summarized in Figure 3.

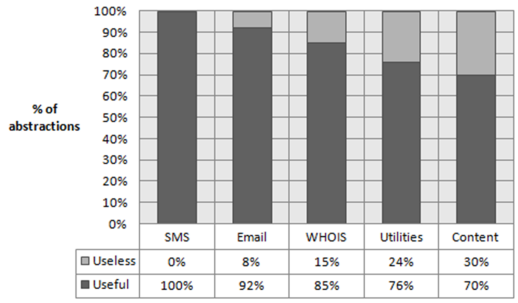
Overall, for all input sets the proposed approach produced relatively high percentages of useful abstractions (ranging from 70% to 100%). However, in certain cases there is also

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

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

[6]The interested reader may find the input sets at http://www.cs.uoi.gr/~dathanas/links/Input_Services_Set.zip

a notable percentage of useless abstractions. This result was expected because the initial design of the algorithm essentially tries to maximize the amount of abstractions that can be constructed from a given set of available services. In particular, any set of services that are more similar to each other, than to the rest of the available services is considered as a candidate for constructing a service abstraction, independently from the degree of similarity of the services. Hence, a straightforward way to reduce the percentage of useless abstractions would be to further constrain the algorithm by setting a threshold for the distance between interfaces that are considered as candidates for extracting a service abstraction. With a threshold value 0.5, the percentage of useless abstractions for the Content category drops to 10%, while for the Utilities category the percentage drops to 12%.



	SMS	Email	WHOIS	Utilities	Content
Useless	0%	8%	15%	24%	30%
Useful	100%	92%	85%	76%	70%

Figure 3: Experimental results.

4. CONCLUSION & FUTURE WORK

In this paper, we proposed an approach for mining service abstractions out of sets of available service descriptions. Our preliminary experimental results were encouraging. In particular, the proposed algorithm produced relatively high percentages of useful abstractions that represent semantically compatible services. Nevertheless, there is certainly room for improvements and further experimentation. As already discussed, it is possible to reduce the percentage of useless abstractions by using a customizable threshold for the distances between service interfaces. More advanced means can be used to assess the similarity of Web service descriptions (employ lexical thesaurus, semantic relations like synonyms, hyponyms, etc.). Moreover, the mining algorithm currently considers only the functional properties of the available services, while it would be interesting to enhance it towards taking into account information related to non-functional properties (e.g., performance, reputation, cost, availability) that may be provided along with the available service descriptions. Another interesting issue would be to account for certain developer preferences and/or business requirements during the mining process. Finally, at this point the mined abstraction hierarchies can be exploited only through browsing. In the near future, we plan to work towards the development of an efficient query engine.

5. ACKNOWLEDGMENTS

This work received funding from the European Community's FP7/2007-2013 under grant agreement number 257178 (project CHOReOS). We would also like to acknowledge the "Equipes Associees" Program of INRIA, supporting the ForeverSOA team.

6. REFERENCES

[1] E. Al-Masri and Q. H. Mahmoud. Discovering Web Services in Search Engines. *IEEE Internet Computing*, 12(3):74–77, 2008.

[2] D. Ardagna and B. Pernici. Adaptive Service Composition in Flexible Processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007.

[3] D. Athanasopoulos, A. Zarras, V. Issarny, and P. Vassiliadis. Hiding Design-Decisions in Service-Oriented Software via Service Abstraction Recovery. Technical Report inria-00491349 - version 2, INRIA, 2010. avail. at http://hal.archives-ouvertes.fr/.

[4] G. Canfora, M. D. Penta, R. Esposito, and M.-L. Villani. A Framework for QoS-Aware Binding and Re-binding of Composite Web Services. *Journal of Systems and Software*, 81:1754–1769, 2008.

[5] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola. Qos-Driven Runtime Adaptation of Service Oriented Architectures. In *Proceedings of the the 7th ACM SIGSOFT ESEC/FSE*, pages 131–140, 2009.

[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] S. Dustdar and M. Treiber. A View Based Analysis on Web Service Registries. *Distributed and Parallel Databases*, 18(2):147–171, 2005.

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

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

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

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

[13] M. Rambold, H. Kasinger, F. Lautenbacher, and B. Bauer. Towards Autonomic Service Discovery - A Survey and Comparison. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*, pages 192–201, 2009.

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

[15] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.