Protocol-Aware Matching of Web Service Interfaces for Adapter Development

Hamid-Reza Motahari-Nezhad*
Hewlett Packard Laboratories
Palo Alto, CA, United States
hamid.motahari@hp.com

Guang Yuan Xu
University of New South Wales
Sydney, Australia
guangyuanxu8@gmail.com

Boualem Benatallah
University of New South Wales
Sydney, Australia
boualem@cse.unsw.edu.au

ABSTRACT

With the rapid growth in the number of online Web services, the problem of service adaptation has received significant attention. In matching and adaptation, the functional description of services including interface and data as well as behavioral descriptions are important. Existing work on matching and adaptation focuses only on one aspect.

In this paper, we present a semi-automated matching approach that considers both service descriptions. We introduce two *protocol-aware* service interface matching algorithms, i.e. *depth-based* interface matching and *iterative reference-based* interface matching. These algorithms refine the results of interface matching by incorporating the ordering constraints imposed by business protocol definitions on service operations. We have implemented a prototype and performed experiments using the specification of synthetic and real-world Web services. Experiments show that the proposed approaches lead to a significant improvement in the quality of matching between services.

Categories and Subject Descriptors

D.2.12 [**Software**]: Software Engineering—*Interoperability*; H.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Design, Experimentation

Keywords

Web Services, Adaptation, Business Process

1. INTRODUCTION

The number of Web services available on the Internet is growing very fast, some of which are functionally equivalent. Functionally equivalent services should be interchangeable [15]. However, such services are often offered using different interface and business protocol specifications. Service interface defines the set of operations that the service provides along with message formats and data types. Business protocol specifies the order in which operations of a service can be invoked [2]. These differences exist despite having standard languages (e.g., WSDL and BPEL) to describe services specifications. Indeed, these languages only provide generic constructs, and using them to define functionally equivalent services by independent teams results in potentially different specifications (may exhibit interface- and protocol-level mismatches) [10, 27].

The problem of Web service matching and adaptation has received significant attention recently [5, 3, 31, 17, 14, 23, 8, 23, 35]. Some approaches (e.g., [3, 14, 23]) identify possible classes of mismatches between service interfaces and protocols and suggest operators or templates for adapter developers to resolve mismatches in each class. Existing automated approaches for service matching and adaptation focus either on the interface-level (e.g., [36, 38, 13, 31]) or the protocol-level (e.g., [3, 8, 37, 28]).

We argue that when matching service specifications, interface and business protocols should not be treated independently. Matching protocol specifications in isolation ignores mismatches at the interface level. And correct matchings at the interface level could be specified more effectively considering the ordering constraints that business protocol definitions impose. For example, Figure 1 shows the operation and protocol definition of two real-world services *XWebCheckout* and *Google Checkout APIs*. Considering only the interface information, in a compatibility checking scenario [6, 2], message AddOrderRequest matches higher to New-Order-Notification in *Google checkout APIs* compared to Place-Order message. Adding the protocol definition information (here mainly the directions of messages –incoming/outgoing– shown by "+"/"-"), the outgoing message AddOrderRequest can only match with incoming Place-Order (remember the compatibility scenario) but not New-Order-Notification which is an outgoing message. As additional evidence, these two messages are in the same depth from the start state of the protocol encoded in state machines.

Taking into account the protocol information makes the service interface matching more precise, productive and efficient. This is due to the elimination of a significant number of false positives compared to when using only XML-based interface information. High false positive ratio is a common issue in XML schema matching [4]. Identifying false positives is a labor-intensive and time-consuming task for users.

As another issue, automated methods for service interface matching (including our previous work [28]) consider one-to-one matching of messages. A common class of matching between interfaces is one-to-many matches (also called message merge/split mismatch) where one message in an inter-

*Most of the work was done when the author was at the University of New South Wales, Australia

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

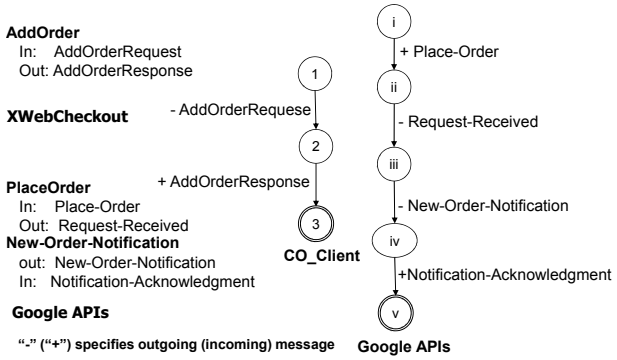


Figure 1: The operations and the corresponding protocols of *CO_Client* (*XWebCheckout*) and *Google checkout APIs* for placing an order.

face is matched to more than one in the other [3, 14]. In this paper, we present a new method for semi-automatic identification of the message merge/split class of interface-level mismatches. We also propose a *protocol-aware* approach for Web service interface matching. Specifically, we introduce the following methods:

- *Message merge/split mismatch identification.* When matching two service interfaces, this method identifies if a given message in one interface is matching (parts of) more than one message in the other WSDL interface. This method extends the static approach for interface matching in our previous work [28] by leveraging XML schema matching algorithms [32, 11, 12]. The innovation is in identifying candidates for message merge/split in the two interfaces.

- *Depth-based interface matching.* This method extends the static approach by incorporating the depth (the number of steps from the initial state of the protocol) in which each message is defined in the business protocol. This is heuristic-based and the intuition behind it is that messages with the same or close depths in the business protocols may have a higher chance of matching. The method reinforces the similarity score of messages appearing in the same (or similar) depth in the business protocol.

- *Iterative reference-based interface matching.* This method incorporates the knowledge of previous matchings to reinforce or penalize the score of not-yet matched (candidate) message pairs. It is iterative, and in each iteration a pair of messages is selected as the best candidate match. This pair is referred to as a *reference* pair for the next iteration. Based on knowledge of the reference pair, we update the similarity scores of other candidate matching message pairs. Updating the scores (penalizing or reinforcing) is performed considering the relative position of a matching candidate compared to the reference pair in the business protocol. The reinforcement process propagates the similarity score of the reference pair into those of their neighboring candidate match that are not in conflict with the reference pair, and penalizes the similarity scores of a message pair that could lead to deadlock in service interactions (is in conflict).

We have experimentally evaluated these methods using specifications of a number of real-world and synthetic Web services. The result of the interface-only matching approach reveals that a major issue is false positives. Reducing the false positive rates is important as the highlighted matchings need to be considered by adapter developers one-by-one. Our protocol-aware interface matching approach proves effective in reducing the rate of false positives significantly and therefore leads to a considerable increase in the quality of matching between service interfaces.

Note that in this paper we focus on matching service interfaces and protocols. For methods related to interface mapping and service adapter development refer to [28, 39]. We have also discussed the feasibility of adapter development for given service specifications and adapter development in [23].

The paper is structured as follows. In Section 2, we present background definitions and problem statement. Section 3 introduces the method for identification of message merge/split mismatches. Section 4 presents our protocol-aware interface matching algorithms. In Section 5, we describe implementation and experimental results. Section 6 discusses the related work, and we conclude and present future work in Section 7.

2. DEFINITIONS AND PROBLEM STATEMENT

2.1 Motivating Example

We present a real-world example[1], which is also used as a running example to illustrate the proposed approach. Let us consider an adaptation task for services in the management of shopping carts. *XWebCheckOut*[2] and *Google Checkout*[3] are commercial checkout services. The two services offer similar functionalities, but through different interfaces and protocols. They provide facilities for sellers to manage the orders that they receive on their own websites. The only major difference between these two services is that *Google Checkout* also provides an administration website for buyers (people who do shopping on sellers' websites). Buyers register their details with *Google* and manage their orders through that website. In *XWebCheckout*, sellers provide administration support for buyers in sellers' websites. Some APIs are provided by *XWebCheckout* to facilitate this task, for which there is no counterpart in *Google APIs*.

XWebCheckout and *Google Checkout* services provide similar APIs for order creation and management, payment processing, and order cancellation. However, there are differences in the interface definitions (message names, number, and types) and how each service expects to exchange messages to fulfill a functionality. For example, Figure 1 shows the protocols of the two services for placing an order. One of the main issues to be addressed for the purpose of adapter development is finding the matching between the interfaces of the two Web services, e.g., to find out that AddOrderRequest *is the corresponding message to* Place-Order in Figure 1. For this purpose, considering only the XML schema definitions of these two services is not enough, as in this case AddOrderRequest would be a better match for New-Order-Notification. Indeed, we need to consider the opera-

[1]We use the same example as of [28] to preserve the consistency of the overall approach
[2]http://www.xwebservices.com/Web_Services/XWebCheckOut/
[3]http://code.google.com/apis/checkout/

tion definition information in the WSDL documents as well as constraints defined on operation invocation (and directions) at the protocol level.

2.2 Background and Definitions

We first revisit the definition of the most common mismatch classes at the service interface level [3]. Let us denote by SP the service provider and SC the service clients to be adapted: (i) *message signature*: message m in SP (corresponding to the request of a certain functionality[4]) has a different name and/or data types in the interface of SC; (ii) *message split/merge*: Message m in SP corresponds to (can be invoked by combining) messages m_1, m_2, ..., m_n in SC, or vice versa;(iii) *missing/extra messages*: One or more messages in SP do not have any correspondence in SC, or vice versa.

To make the paper self-contained, we give the definition of the interface and interface mappings first presented in [28]. The definition of interface I of a Web service S, denoted by I_s, is a simple formalization of WSDL:

Definition 2.1 (Web service interface)**.** *A Web service interface I_s is a triplet $P = (D, M, O)$, where D is the set of (XML) data types of the service, O is the set of operations supported by the service, and M is the set of messages exchanged as part of operation invocations, in which*

- *a message m has i parts ($i \geq 1$), represented as $m =< d_1, d_2, ..., d_i >$, $m \in M, d_j \in D, 1 \leq j \leq i$*
- *$o = \langle m_{req}, m_{res}, m_f \rangle$, that is, $o \in O$ is an operation associated to at least a request message m_{req} or to a response message m_{res} (or both) and optionally a fault message m_f.*

For simplicity we write $m \in I_s$ equivalent to $m \in M_s$. We define the mapping between two Web service interfaces I_c and I_s as follows:

Definition 2.2 (Interface Mapping)**.** *Given interfaces $I_s = (D_s, M_s, O_s)$ of service S_s and $I_c = (D_c, M_c, O_c)$ of service S_c, an interface mapping $IM_{<s,c>}$ from I_s to I_c is a set of functions such that: $m \leftarrow func(X)$, $m \in M_s$ and where the input X is either a set of messages $\{m' | m' \in M_c\}$ or a set of constant values.*

The interface mapping $IM_{<s,c>}$ may contain more than one mapping function for a given message $m \in I_s$, or may not contain any function for another message $m' \in I_s$. This definition allows for specifying one to one $(1-1)$ mappings (to model message signature mismatch) and one to many $(1-n)$ mappings (modeling message split/merge mismatch).

We use P_s and P_c to denote the protocol definitions of provider S_s and a client service S_c, respectively. We adopt finite state machines (FSM) as the modeling formalism for business protocols [2].

2.3 Problem Statement

The problem we tackle in this paper can be stated as that of providing semi-automated support for identifying matchings between service interfaces for the purpose of adapter development starting from service interfaces, I_s and I_c, and protocol definitions, P_s and P_c of services S and C. The main challenges of tackling this problem include providing efficient and effective protocol-aware approaches for matching of Web service interfaces, and identification of correspondences and mismatches at the interface level. We define the problem as follows:

Problem 2.1 (Service Interface Matching)**.** *Interface matching for I_s to I_c refers to identifying the correspondences between messages in I_s and I_c, i.e., the set $X \in M_c$ of parameters of the function $func(X)$ for generating $m \in M_s$ in function $m \leftarrow func(X)$, and vice versa for I_c to I_s.*

Note that during the interface mapping, the body of the interface mapping functions $m \leftarrow func(X)$ are implemented. The identification of the set X, i.e., interface matching, is the most important step in specifying $func(X)$. In the following, we present novel methods for finding the set of parameters for $func(X)$.

3. STATIC INTERFACE MATCHING: MESSAGE MERGE/SPLIT IDENTIFICATION

WSDL interface definitions are XML documents, and the type of data exchanged by messages are defined using XML schema. We presented a method for $1-1$ matching of (WSDL) service interfaces in [28] by leveraging XML schema matching algorithms [32, 33]. This approach considers the structure of WSDL documents, i.e. the operations and their input/output definitions as well as XML schema definitions. It extracts the schema definition for each input/output message. Then, it matches the XML schema of pair-wise messages in two interfaces and generates a *similarity score* between messages. In addition to the XML schema of messages, we take into account message names, and whether a message is input or output of an operation as it reduces the number of required pair-wise message matchings.

In this paper, we propose a new method for the identification of mismatches of type split/merge $(1-n)$. In $1-n$ matching, some schema elements of a message $m \in I_s(I_c)$ are matched with elements $m' \in I_c(I_s)$ and some other elements of m are matched with some elements in $m'' \in I_c(I_s)$, etc. Messages m' and m'' are called *component* messages of m denoted by $m' \triangleright m$ and $m'' \triangleright m$. In general, we may have many-to-many $(p-q)$ matching in which elements of p messages in one interface are matched with the elements of q messages in the other interface. Here, we focus on the identification of component messages of a given message for message split/merge mismatch class, i.e., $1-n$ matching.

Algorithm 1 shows the proposed method. Given interfaces I_s and I_c, we look for any messages $m_1, m_2, ..., m_n \in I_s$ such that $m_1, m_2, ..., m_n \triangleright m'$, $m' \in I_c$ (or reversely from I_c in I_s). Let $m_1 \in I_s$ be a candidate message that we want to see whether $m_1 \triangleright m_2 \in I_c$. We apply two heuristic criteria to find component messages. The first criterion makes sure that the similarity score of m_1 and m_2 (denoted as $S(m_1, m_2)$) is above a threshold t_1 so that there is a likelihood of matching between two messages. The threshold t_1 is specified according to the matching algorithm that is used in experiments (see Section 5.2). The second criterion makes sure that a significant number of elements in m_2 (a default heuristic is more than half, but is configurable) are matched with those of m_1. If there are more than two messages that are identified as components of m_1, then it is told that there is a mismatch of type split/merge. Note that the

[4]Receiving (sending) a message corresponds to invoking an operation (its reply, respectively).

parameter *Algo* in similarity score $S(m_1, m_2, Algo)$ specifies the schema matching algorithm used in $1-1$ matching (see Section 5.2 for discussion of options).

Algorithm 1 Split/Merge Mismatch Identification

Require: WSDL interface I_s, WSDL interface I_c
Ensure: ComponentList *comList*
1: $comList \leftarrow \emptyset$
2: **for** each message $m_1 \in I_1$ **do**
3: $comList(m_1) \leftarrow \emptyset$
4: **for** each message $m_2 \in I_2$ **do**
5: **if** $S(m_1, m_2, Algo) > t_1 \&\&$ $num(MatchingElement(m_1, m_2))/num(Elements(m2)) > c_1$ **then**
6: $Elements(m1) \leftarrow$ $Elements(m1)$ - $MatchingElement(m_1, m_2)$
7: $comList(m_1) \leftarrow comList(m_1) \cup \{m2\}$
8: **end if**
9: **end for**
10: **if** getSize($comList(m_1)$) ≥ 2 **then**
11: $comList \leftarrow comList \cup \{(m_1, comList(m_1))\}$
12: **end if**
13: **end for**

Note that we consider matching of service messages in a pair-wise manner. The granularity of matching message parts is specified by XML schema matching algorithms.

4. PROTOCOL-AWARE INTERFACE MATCHING

The protocol-aware approach incorporates protocol level information into the interface matching process. We introduce the following two methods:

- *Depth-based* approach for improving the static matching-based similarity score for messages with a same/close depth (the distance of the transition labeled with the message from the initial state) in a protocol, and
- *Iterative reference-based* approach for propagating the similarity score of messages into those of their neighbor messages in the protocols, and also using already matched message pairs as *references* to reinforce or penalize the similarity scores of messages before and after this pair of messages in the protocol. It builds on the depth-based method by also considering the depth information in computing the similarity score, and extends the depth-based method in how scores are computed.

4.1 Depth-based approach

The intuition behind the depth-based approach is that messages with similar depths in the two protocols P_c and P_s are more likely to match. This heuristic holds when there are no interface-level mismatches between protocols and the two business protocols are compatible (see [2, 6] for protocol compatibility discussion). We expect this heuristic to hold to a large degree when the protocols belong to functionally compatible services, for which adapter development is a viable solution compared to developing a new client from scratch (see service adaptation feasibility discussion in [23]).

As an example, Figure 2 shows simplified protocols of *CO_Client* and *Google Checkout APIs*. Messages AddOrder and PlaceOrder which are in the same depth of 1 are more likely to be a correct match. It is useful to consider the direction of messages, i.e., incoming (+) or outgoing (-) in identification of depth, as well.

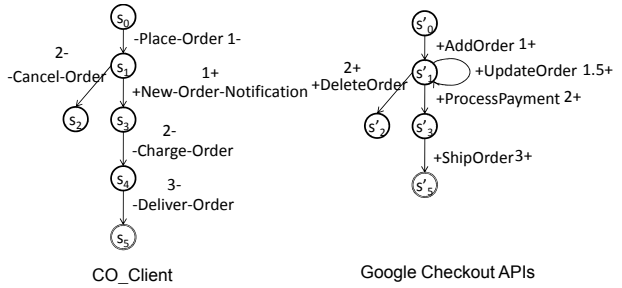


Figure 2: The simplified protocol specifications of *CO_Client* and *Google checkout APIs* associated with respective depth numbering

A protocol may have loops, i.e., transitions recurring back to states closer to the initial state. To avoid going into infinite loops for identifying the depth of a message associated to such transitions, we propose to first normalize the protocol into a *protocol tree* in which only self-loops are allowed. The normalization process involves traversing each transition and checking whether the outgoing transitions for the target state of the current transitions are already labeled with a smaller depth number. If any of the outgoing transitions matched this criterion, then the normalization process does not continue this path of the protocol. Then, the depth of a message is specified by the number of transitions to be traversed from the initial state s_0 to the message m in this tree-like representation of the protocol.

In this process, messages of each direction are numbered separately. For instance, the message -Place-Order gets the number 1− and the message +New-Order-Notification the number 1+. These numbers are relative but do not correspond to the exact depth of a message from the initial state. In case of self-loop in P (e.g., see message +UpdateOrder in Figure 2), we associate the number 1.5+ to show that it is a self-loop (but not the depth of 2).

Given the depth of messages in the protocol, we update the $1-1$ similarity scores of messages in two interfaces. This process improves the score of each matching considering the depth information as follows. Let $m_1 \in I_1$ and $m_2 \in I_2$, and $S(m_1, m_2)$ be the similarity score of messages m_1 and m_2[5]. The improvement weight for the score in the depth-based approach is shown in Equation 1.

$$scale \leftarrow \frac{maxdepth - |depth(m_1) - depth(m_2)|}{maxdepth}. \quad (1)$$

The new score $S'(m_1, m_2)$ is computed as

$$S'(m_1, m_2) = S(m_1, m_2) \times scale. \quad (2)$$

This ensures that messages with closer depth will be scaled higher in comparison with messages that are further apart from each other. Note that *maxdepth* is a constant that defines the maximum depth obtainable in the protocol for each direction. This approach enhances the accuracy of the matching compared to the static approach, as verified by experiments (see Section 5.2).

[5]Note that $I_1 = I_s(I_c)$ and $I_2 = I_c(I_s)$

4.2 Iterative reference-based approach

The iterative reference-based approach improves the depth-based approach in two ways: (i) considering additional protocol information including the path in which messages are located in the protocol tree, as well as previous matchings for identifying the similarity, and (ii) relaxing the implicit assumption in the depth-based approach, i.e., that of the similarity of the structure of protocol trees for functionally equivalent services at a global level. The relaxation is achieved by by allowing the initial matching of message pairs at any levels of the protocol trees.

This method is iterative and in each iteration a pair of messages is selected as the best candidate match. Assuming it is a correct match (a user could be involved to verify the correctness of the match), this pair is selected as a reference used to update the similarity scores of other yet un-matched message pairs. The other scores are either reinforced or penalized.

As a pre-processing step, each protocol tree is decomposed into distinct paths. The best candidate pair of messages is considered a *reference matching pair* (or reference pair for short) in the same path-pair (see strategies for selecting the references later). For instance, Figure 3(a) shows two paths from protocols P_s and P_c, respectively. If message pair $-c$ and $+c'$ are the best candidate match, we select them as a reference pair. A reference pair could be used to reinforce or penalize the other matching candidates in the same path-pair. The reference-based method refines the $1-1$ message similarity scores by taking into account the following:

- *Depth-based score improvement.* Similar to depth-based, messages with similar depth in the two protocols P_1 and P_2 are considered more likely to match (see Section 4.1).
- *Propagation of similarities to neighbors of a matching message pair.* If $-c \in I_1$ and $+c' \in I_2$ are selected as a reference pair, then their neighbor message before (after) the reference point are more likely to have a matching to others before (after) $+c$ (respectively, $-c'$) in those paths. For instance, if c and c' represent the matching invoice messages, then it is likely the messages before these messages, which are about quote and ordering, and also messages that are after these in the protocols, e.g., about payment and shipping, are more likely to match with each other, respectively. The reinforcement is done using a *rate* that controls the rate at which the reinforcement decays depending on how far the matching messages are from the reference message pair.
- *Penalizing conflicting matching candidates crossing reference pairs.* Given reference pair $+c \in I_1$ and $-c' \in I_2$ the candidate matching pair $-f \in I_1$ and $+j' \in I_2$ (shown as crossing the solid reference line in Figure 3(b)) is called a *conflicting* match. This is because $-f$ (an outgoing message) with a bigger depth than $+j'$ (an incoming message) leads to a deadlock[6] in the interaction of two services in case this matching is allowed. Therefore, the similarity score of a conflicting matching pair is penalized.

[6]services are mutually waiting indefinitely to receive messages from each other

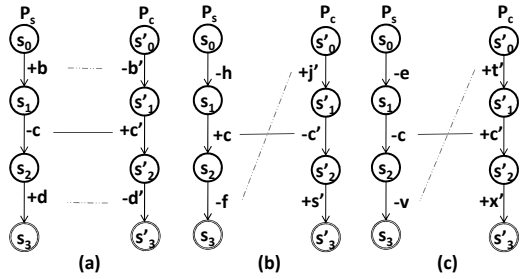


Figure 3: Two paths in protocols P_s and P_s. The reference pair is shown with a solid line.

Propagation of similarity scores to neighbors. The propagation of similarities to neighbors of a reference pair is achieved through reinforcing the similarity scores of pairs that are not in conflict with the reference pair. In Figure 3(a), the matching of $+b$ and $-b'$, as well as $+d$ and $-'d$ are reinforced. However, the similarity scores of b and d' and b' and d are not reinforced as their matching lines (if drawn) cross the reference line. This is logical as P_s expects to receive b before it sends c. On the other hand, considering the matching between b' and d, P_c sends b' before waiting for c', and P_s is waiting to receive d after sending c. There are no conflicts between the exchange of these messages and so these two can be a possible match. However, we do not reinforce or penalize its similarity score.

Penalizing scores of conflicting matching pairs. As mentioned above, a conflicting matching pair should be penalized to avoid potential deadlocks in service interactions according to the protocols. A conflicting matching is identified among all candidate matching pairs whose line crosses the reference line by the condition that the message with the "+" sign (say m_1) is before the reference message m_{ref1} (with "-" sign") in the path and the message with the "-" sign (say m_2) is after the reference message m_{ref2} (with "+" sign) in the path.

For example, in Figure 3(b) the matching of $+j'$ and $-f$ (shown with a dashed line) satisfies this condition and therefore is penalized. Note that not all crossing matching candidates are penalized. For instance, in Figure 3(c) the matching candidate pair $+t'$ and $-v$ are not penalized although $+t'$ is in a smaller depth compared to $-v$. This is because of the direction of $-c$ in the protocol path P_s. Indeed, this case identifies mismatch of type unspecified reception, which can be handled as a sub-class of ordering mismatch patterns (a protocol-level mismatch) in the adapter [3, 28]. Similarly, the similarity score of $-e$ with $+x'$ in Figure 3(c) is not penalized. In general, if we have crossing matches for which the message with "-" sign is above the reference message pair and "+" is below the reference message, it is not penalized (as an adapter can receive the message with "-" sign and store it in the adapter for future use).

The success of this approach is directly related to the selection of reference pairs, and whether they are correct. We discuss this aspect next.

Reference message selection. We introduce two methods for the selection of a reference pair: (i) *Automated reference selection*: In each iteration the pair with the highest $1-1$ similarity score between the two interfaces is selected as the reference and the similarity scores of others are updated, and (ii) *User-driven reference selection*: In each iteration,

the best candidate matching pair is selected and presented to the user for his/her feedback on the correctness of the match as well as reference selection. Based on the user feedback, the similarity scores are updated and the next reference is selected. In the automated reference-selection the success of this approach depends on the quality of the matching algorithm. The user-driven reference selection greatly improves the accuracy and effectiveness of matchings (see Section 5.2).

Algorithm 2 Iterative Reference-based Interface Matching

Require: $mList_1$, $mList_2$
Ensure: $mrList$
1: $mrList \leftarrow$ depth-basedMatching($mList_1$, $mList_2$)
2: $mp \leftarrow$ selectReferencePoint($mList_1$, $mList_2$, $mrList$)
3: **while** $mp \neq \emptyset$ **do**
4: $m_{ref1} \leftarrow mp.m_1$, $m_{ref2} \leftarrow mp.m_2$
5: **for** each $m_1 \in mList_1$ **do**
6: **for** each $m_2 \in mList_2$ **do**
7: $mr \leftarrow$ getMatchResult(m_1, m_2, $mrList$)
8: $diff \leftarrow max(abs(m_{ref1}.depth\text{-}m_1.depth), abs(m_{ref2}.depth\text{-}m_2.depth))$
9: $rate \leftarrow 1 + (maxdepth - diff)/maxdepth$
10: **if** $[(m1, m2)$ cross $(m_{ref1}, m_{ref2})]$ and $conflict$ **then**
11: $mr.score \leftarrow mr.score \times 1/scale$
12: **else if** m_1 (m_2) neighbor of $m_{ref1}(m_{ref1})$ **then**
13: $mr.score \leftarrow mr.score \times rate$
14: **end if**
15: **end for**
16: **end for**
17: $mp \leftarrow$ selectReferencePoint(I_1, I_2, $mrList$)
18: **end while**

Algorithm 2 summarizes the iterative reference-based approach. The short forms of *mList* refer to list of messages (e.g., $mList_1$ refers to the list of messages in I_1), *mrList* refers to match result list, *mp* stands for message pair, and *mr* stands for match result. In this algorithm, static match scores are computed and reinforced using depth-based approach (line 1). Then, a reference pair is specified using one of the approaches explained above (line 2). Next, the scores for message pairs leading to deadlock are penalized by multiplying the similarity score by the inverse of *scale* (line 11), which is computed according to Equation 1. If the pair of messages are neighbors of reference pairs, their score is reinforced proportionally to the difference of their depth with those of messages in the reference pair (line 13). This is done by multiplying their similarity score by the reinforcement *rate* (computed in line 9). Afterwards, the algorithm looks for the next reference pair, if any. The algorithm terminates when there are no further reference pairs to be selected.

5. IMPLEMENTATION AND EXPERIMENTS

5.1 Implementation and Dataset

Implementation. The approaches presented in this paper have been implemented in Java 5.0 using Eclipse 3.2 as the programming IDE. The static interface matching approach in the interface matching component is implemented on top of the OntoBuilder[7] library of ontology matching algorithms [29, 12, 18]. OntoBuilder is an automatic schema matching tool based on ontological constructs. An ontology consists of terms which are linked together with a parent or child relationship. OntoBuilder provides Java APIs that allow programmatic access to its capabilities.

We have extracted individual messages of WSDL documents along with their XML schema and built new schemas to include WSDL contextual information as described in Section 3. We then convert these schemas into ontologies in OntoBuilder and match them using various built-in ontology matching algorithms, e.g., term-based, graph-based, precedence, similarity flooding algorithm, and a combined algorithm using all these algorithms at the same time. In $1 - n$ interface matching we use the result of XML schema matching from OntoBuilder. We have also compared our algorithms with the approach that uses only schema matching algorithms (in our implementation from OntoBuilder) for service interface matching (reported below).

Dataset. We have evaluated the proposed methods in both synthetic and real-world scenarios. The WSDL documents used for the evaluation in the real-world scenarios are obtained from real world web services in two categories: purchase order and shopping card management services (*Google Checkout*[8], *XWebCheckout V2*[9], *Moon Purchase Order Management Service*[10], *Amazon Web Service*[11] and *Amazon E-commerce Service*[12]), and payment services (*PayPal Web Service*[13], *PaymentExpress Web Service*[14], *Amazon Flexible Payments Service*[15]).

Some of the WSDL documents have also been modified by adding operations and messages in order to evaluate the approach with protocols with various degrees of complexity. The business protocols are constructed using the information from documentation and implementation guides for those real world web services. They represent various scenarios such as ones with simple protocols having a few messages (each 3-5 messages with large schemas), complex protocols involving many messages (each up to 15 messages with large schemas), protocols with self-loops, protocols with loops back to states closer to the initial states of protocols and protocols with multiple paths.

The synthetic service specifications are created by manipulating the real-world service specifications to validate the effectiveness of protocol-aware approaches in scenarios not covered by real-world examples. In particular, we validate the effectiveness and usage of protocol-aware approaches in cases that $1 - 1$ matching (syntactical) results are correct (where only protocol-level mismatches are present). We have used a desktop P4 Intel CPU 2.8GHz with 4 GB of RAM to conduct the experiments.

5.2 Experiments

5.2.1 Evaluation criteria

We use the following criteria to evaluate the efficiency and the accuracy of the proposed approaches with each other and existing work. For the evaluation of accuracy, we use the well-known measures of *precision* and *recall* [34]. Precision

[7] http://iew3.technion.ac.il/OntoBuilder/

[8] http://code.google.com/apis/checkout
[9] http://www.xwebservices.com/Web_Services/XWebCheckOut
[10] http://sws-challenge.org/services/OMService?wsdl
[11] http://soap.amazon.com/schemas2/AmazonWebServices.wsdl
[12] http://webservices.amazon.com/AWSECommerceService
[13] http://www.paypal.com/wsdl/PayPalSvc.wsdl
[14] https://www.paymentexpress.com/WS/PXWS.asmx?WSDL
[15] https://fps.amazonaws.com/doc/2007-01-08/AmazonFPS.wsdl

measures the quality of the matching results, and is defined by the ratio of the correct matches, in terms of message pairs in the two interfaces, to the total number of matches found. Recall measures coverage of the matching results, and is defined by the ratio of the correct message pairs matched to the total number of all correct matches of message pairs that should be found. These two definitions are summarized in Equation 3 and Equation 4:

$$Precision = \frac{number\ of\ correct\ message\ pairs\ matched}{total\ number\ of\ message\ pairs\ matched} \quad (3)$$

$$Recall = \frac{number\ of\ correct\ message\ pairs\ matched}{total\ number\ of\ correct\ message\ pairs\ in\ the\ two\ interfaces} \quad (4)$$

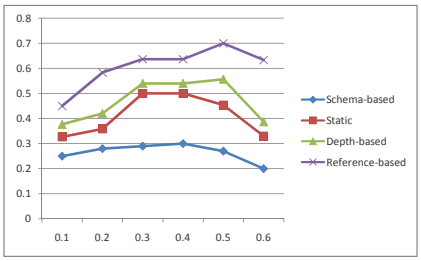
For an approach to be effective, it should achieve a high precision and high recall. However, in reality these two metrics tend to be inversely related [34]. This means that the improvements in precision come at a cost of reduction in recall, and vice versa. For identifying the set of correct matching between message pairs of two given interfaces, we rely on the judgment of human users. To evaluate the efficiency of the matching approaches, we measure and compare the time needed to perform the matching of WSDL documents using different approaches.

5.2.2 Evaluation Results in Real-world Scenarios

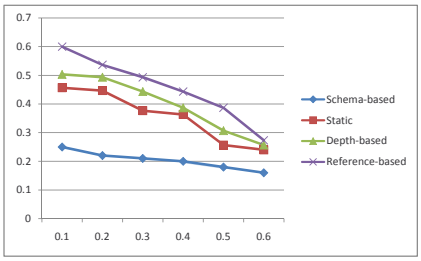
In the evaluation, we refer to the protocol-aware interface matching approach as *reference-based*, the depth-based approach as *depth-based* and the static $1-1$ matching algorithm as *static*. We also compare this result with those of only using a schema (ontology) matching algorithm (without including WSDL context information such as operations) but only matching the whole XML schema of two services at once. We refer to this approach as *schema-based*. We first provide the comparison of the above approaches in one representative scenario and then analyze results in others.

Google Checkout and XWebCheckout V2. Figure 4 shows the result of applying these algorithms averaged on three versions of *Google Checkout* and *XWebCheckout V2* specifications having simple protocols to a complex one that includes 9 additional operations, taken from other similar services, regarding the purchase order management. The static schema matching approach, reported here, uses the combined schema matching algorithm in OntoBuilder (with default settings) applying term, graph and precedence evidences for ontology matching. The chart shows the results for thresholds from 0.1 to 0.6 with an interval of 0.1. The threshold is applied on the similarity scores, obtained from the schema matching approach in the static approach, and also improved scores in the proposed protocol-aware approaches. It is used to decide if a pair of messages match.

As can be see from the charts in Figure 4, the general trend of the precision chart is that the precision starts increasing for threshold 0.1 to 0.4. The precision of the schema based approach is computed using the whole schema of the two services in matching. The precisions for the schema-based and static approaches start declining from the threshold 0.4, and for others from 0.5. On the other hand, the general trend of the recall is consistent and decreasing as the threshold increases. Note that, in general, the precision and recall of static matching are low. This is due to the heterogeneity



(a) Precision: X-Axis shows the threshold, and Y-Axis shows the precision



(b) Recall: X-Axis shows the threshold, and Y-Axis shows the recall

Figure 4: The evaluation of the proposed approaches on *Google Checkout* and *XWebCheckout V2*

of the details of XML schemas defined by *Google Checkout APIs* and *XWebCheckout*.

The charts show that using the static approach to divide the schemas of services into those of messages leads to always achieving a better precision and recall compared to the schema-based approach. The precision of the static approach is increasing for thresholds 0.1 to 0.4, and its recall is constantly decreasing for this range of threshold, but always demonstrates a superior performance compared to schema-based approach. In addition, the depth-based approach improves the results of static approach for all thresholds as it boosts the similarity scores for the ones that are more likely to match. Finally, the reference-based approach achieves a better precision than others due to disallowing deadlock cases, and it achieves a superior recall due to the propagation of similarities to the neighbors thus allowing matching message pairs, which are correct, to achieve higher scores. In general, for the higher thresholds (up to 0.4), the precision increases as the number of matching pairs that are returned also decreases, among which the correct ones are present. In this evaluation, each reference pair is verified by the user meaning that the highest matching pair are suggested to the user as the reference consecutively until a correct pair is selected, then the propagation of similarities is performed.

The un-expected decline in the precision of the static approach after the threshold of 0.4 is because the similarity score values for many correct matches fall below this threshold value. Similarly, the overall precision for other approaches also decreases for thresholds higher than 0.5. This is not expected, as we expect the precision to increase as the threshold increases. The reduction in the precision in this case is due to the fact that the highest scored matching obtained from the static approach are not the correct ones. Hence, when the threshold increases the correct matches are removed from the set of resultant matching. However,

Table 1: The average execution time of different approaches using OntoBuilder for matching schemas of services

Approach	Time
Schema-based	17 min 34 sec
Static	40.5 sec
Depth-based	42.2 sec
Reference-based	45.7 sec

as we see the depth-based approach, and also the iterative reference-based approach, play the role of improving the scores for the correct matching pairs to keep them in the set of resultant matchings for threshold values of 0.4 and above.

Threshold selection. Note that the selection of threshold depends on the algorithm and the tool used for matching of schemas, and also the schema of the services to be matched. There has been some work in the area of schema matching to find parameters of matching algorithms based on the problem at hand (e.g., see [24]). As can be seen from the results, selecting a threshold closer to the lower end of the spectrum may result in lower precision, but, it achieves a higher recall. It has been reported in [20] that in information retrieval tasks users would tolerate a decrease in the precision if it can bring about a comparable increase in recall. This observation should be used in selecting the threshold.

Efficiency. Evaluation of the efficiency of the approach shows that applying the proposed static interface matching significantly reduces the time needed to match the schemas of the interfaces of two services (See Table 1). While matching the whole schema of *Google Checkout* and *XWebCheckout* using *OntoBuilder* takes more than 17 minutes for OntoBuilder, the static approach takes 40.5 seconds, the depth-based approach 42.2 seconds and finally reference-based approach (when all steps are performed automated) takes on average 45.7 seconds. The remarkable improvement in the time stems from two factors: the small size of fragments of schema, corresponding to those of messages, and matching not all pairs of messages but some of them.

Results on other datasets. We have performed experiments using the specifications of other Web services, i.e., among payment Web services (*PayPal* and *PaymentExpress*, *AmazonFPS* and *PayPal*), purchase order and shopping card management Web services (*Google Checkout* and *Moon's purchase order management service*, *Amazon Web service* and *Amazon E-commerce service*). Here, we discuss our observations and lessons learned from these experiments.

The total time taken for the execution of the static and also protocol-aware approaches for interface matching is significantly less than that of using the schema-based approach on the whole WSDL schema definitions of services, confirming the observation that can be made in Table 1.

The proposed approaches improve the accuracy compared to the schema-based approach. On average, the reference-based approach, based on results obtained on real-world datasets, demonstrates superior performance in comparison with the depth-based and static approaches, and all performed better than applying the schema-based approaches. The results of the iterative reference-based approach are superior when there are conflicting mismatches (leading to deadlock) defined in the protocols. In other cases, it performs slightly better than depth-based approaches due to the propagation of scores to the neighbor messages in the protocol but often the results are comparable.

As for the recall results, a consistent decreasing trend is observed for all three approaches as the threshold value increases, as expected. The protocol-aware approaches demonstrate a higher recall, on average, compared with the static and also schema-based approaches. In summary, comparing the results from the protocol-aware approaches with the static approach also reveals that as protocols become more complex the benefits of using protocol-aware approaches are more significant.

Refined reference-based approach. We performed experiments to select the reference pair automatically by taking the matching pair with the highest score. In most cases, the pair with the highest score, especially in the first iteration, does not represent a correct match. This negatively affects the precision and recall of the iterative reference-based approach. In these cases, we have performed experiments with a variation of the reference-based approach, in which it does not penalize the matching leading to deadlock, since we are not sure if the selected matching is correct. Comparing this approach with static and depth-based approach reveals that it improves the recall but achieves a precision slightly lower than depth-based approach. This is because a higher number of message pairs are returned, and some of them are not correct.

5.2.3 Evaluation in synthetic scenarios

The aim of the synthetic scenarios is to evaluate the effectiveness of the protocol-aware interface matching when the matching between interfaces is correct according to the information at the interface level. We report on using two pairs of synthetic order management protocols to evaluate the approach. In particular, the first scenario is represented by a pair of protocols that involve matching pairs leading to deadlock cases, similar to cases presented in Figure 3. The precision of the static algorithm in this case is 83%. Applying the depth-based approach does not change the precision. This is because the depth-based approach only enhances the similarity scores of messages at same level. However, the iterative reference-based approach achieves a precision of 100%. This is because it does not allow matching leading to deadlock cases.

The second scenario consists of a pair of protocols involving a deadlock case and also a message matching in a path of protocol matching two messages in two different paths, only one of them correct. In this case, the precision of the static and depth-based approaches is 86%. This is improved to 93% for the iterative reference-based addressing the deadlock case. However, the iterative reference-based approach is not sufficient in finding the matchings to different paths showing the need for complete protocol-interaction simulation, a complementary approach that was proposed in our previous work [28] for identifying protocol-level mismatches.

6. RELATED WORK

The problem of adapting interaction models in software has been studied in different contexts, and more notably in the area of software components integration (e.g., [39, 7, 22]) and also recently in Web services [3, 17, 14, 23, 8, 35]. In the following, we discuss the related work in three categories, i.e., interface matching, behavioral matching as well as diagram matching.

Interface matching. At the interface level there are four main classes of approaches in the prior art. The approaches in the first class focus on finding the similarities between a given function (operation) signature to others in a repository of software components [40, 41] or Web services [13, 36] specifications. In [40] authors propose a method to compare the signature of software components and identify their matching (and identifying a measure of similarities) considering the parameter name, parameter type, parameter order, etc. This corresponds to $1-1$ matching in our work. In addition, service interfaces (described in WSDL and XML schema) are more expressive than software signatures. In finding similar services to a given query or operation from a service repository (e.g., [13, 36]), the objective is not to find the exact matching between schema elements of messages but to find a measure of their similarity typically based on information retrieval techniques. In addition, their methods rely on learning and statistical analysis (e.g., clustering) of existing service interface specifications. In contrast, the aim in our work is to find exact matching of a given pair of service specifications (WSDL).

The second class of prior art proposes approaches for adapting a service WSDL interface to incompatible clients, e.g., [17, 31]. In [31] authors assume that interfaces of all services that provide a similar functionality are derived from a common base interface using a number of derivation operators that allow for adding or removing parameters to operations. However, the operation names and other aspects of the service interface remain the same. In [17], the author proposes defining service views on top of WSDL interfaces by altering WSDL interfaces to enable interactions with incompatible services. However, no automatic support for the generation of views is proposed. We focus on semi-automatic protocol-aware interface matching for service adaptation.

As the third class there are ontology-based approaches for service interface matching (e.g., [1, 30, 18]). These works correspond to $1-1$ matching in our approach. In addition, they do not consider protocol constraints. We also build on top of the ontology matching approach offered by the OntoBuilder tool [29, 18, 12] to perform $1-1$ matching of messages in the interfaces of Web services.

Finally, as the last class, commercial products, e.g., IBM WebSphere Integration Developer[16], BEA WebLogic Integration [17] or Microsoft Biztalk[18] also have integrated existing methods for schema matching. Hence, they share the same limitations of schema matching approaches compared to the approaches presented in this paper for service specifications matching.

Behavioral and diagram matching. There are many approaches for behavioral matching of software components (e.g., [7, 21, 41, 9, 16]) or services (e.g., [37, 2, 6]). They are focused on matching business process and protocol specifications in the absence of interface mismatches. Our previous work [28] is the only one that reports matching service interface as well as service protocol specification. However, service interfaces are matched using a static approach $(1-1)$. Tan et al. [35] present a protocol-level mediation method and assume that the interface-level mismatches are identified and given by the developer. In this paper, we are considering matching of interfaces in the context of service adaptation, rather than trying to find a measure of similarity of one protocol with others in a repository, which is the focus of another complementary line of research (e.g. [19]).

The approach presented in this paper builds on and extends our previous work in [3, 28]. In particular, in [28], we presented a method to identify one-to-one matching of service messages at the interface level, and one-to-many matching is only sketched. In this paper, we proposed a new method for identifying one-to-many (merge/split) mismatches. More importantly, we presented a *protocol-aware* approach for Web service interface matching, which is complementary to our previous work on bi-simulation of interaction of business protocols for service adaptation [28].

There is another class of related work for matching software architectural diagrams as well as software behavior models (e.g., [25, 26]). These works are focused on finding similarities between diagrams. In this paper, we focus on matching the XML-based interfaces of Web services (considering protocol constraints), which are much more expressive than labels on diagram transitions. The approach presented in this paper may be useful in finding more complex relationships between diagrams as well (e.g., method for split/merge mismatch identification). However, some other aspects and methods such as deadlock case elimination may not be needed in the context of diagram matching.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have investigated the problem of identification of mismatches between interfaces of Web services. The innovative contributions of the proposed approaches lie in providing a method for identification of the split/merge class of interface mismatches and a semi-automated, protocol-aware approach for identification of interface-level mismatches that result in identifying parameters of mapping functions that resolve those mismatches. We used and extended approaches in schema (ontology) matching for static matching of service interfaces to identify split/merge mismatches. In addition, we have proposed depth-based and also iterative reference-based approaches that incorporate the protocol information during the interface matching.

We have implemented the approach in a prototype tool and performed extensive experiments using both synthetic and real-world service interface and business protocol specifications. The result shows that these approaches considerably improve the effectiveness and the accuracy of matching results. The work presented in this paper complements our previous work and we believe that together they play a significant role in reducing the efforts for adapter development in Web services.

As future directions in this area we are considering extending the matching algorithms to identify other classes of mismatches between services interfaces. We are also considering incorporating interface and protocol matching approaches in the composition of Web services as current approaches often do not consider heterogeneities of service specifications while composing services.

8. REFERENCES

[1] R. Akkiraju, A.-A. Ivan, R. Goodwin, S. Goh, and J. Lee. Semantic tools for web services. In *Emerging Technologies Toolkit (ETTK), IBM AlphaWorks.* http://www.alphaworks.ibm.com/tech/wssem, 2006.

[16]http://www.ibm.com/software/integration/wid
[17]http://www.oracle.com/bea/index.html
[18]http://www.microsoft.com/biztalk

[2] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *DKE J.*, 58(3):327–357, 2006.

[3] B. Benatallah and et al. Developing adapters for web services integration. In *In Proc. CAiSE*, pages 415–429, 2005.

[4] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental schema matching. In *VLDB'06*, pages 1167–1170, 2006.

[5] D. Beyer, A. Chakrabarti, and T. Henzinger. Web service interfaces. In *Proc. the 14th World Wide Web Conference (WWW 2005)*, pages 148–159, 2005.

[6] L. Bordeaux and et al. When are two web services compatible? In *Proc. Workshop TES in Conjunction with VLDB*, pages 15–28, 2004.

[7] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *J. Systems and Software*, 74(1):45–54, 2005.

[8] A. Brogi and R. Popescu. Automated generation of bpel adapters. In *Proc. ICSOC*, pages 27–39, 2006.

[9] C. Canal and et al. Adding roles to CORBA objects. *IEEE Trans. Software Eng.*, 29(3):242–260, 2003.

[10] K. Chari and S. Seshadri. Demystifying integration. *Commun. ACM*, 47(7):58–63, 2004.

[11] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. VLDB*, pages 610–621, 2002.

[12] C. Domshlak, A. Gal, and H. Roitman. Rank aggregation for automatic schema matching. *IEEE TKDE*, 19(4):538–553, 2007.

[13] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proc. VLDB*, pages 372–383, 2004.

[14] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *Proc. BPM*, pages 65–80, 2006.

[15] T. Erl. *Service-Oriented Architecture: Concepts, Technology and Design*. Prentice Hall PTR, 2005.

[16] A. Faras and Y. Gueheneuc. On the coherence of component protocols. *Electr. Notes Theoritical Computer Science*, 82(5), 2003.

[17] M. Fuchs. Adapting web services in a heterogeneous environment. In *Proc. ICWS*, 2004.

[18] A. Gal and et al. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB J.*, 14(1):50–67, 2005.

[19] D. Grigori and et al. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Inf. Syst.*, 33(7-8):681–698, 2008.

[20] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Proc. Int'l Conf. Requirements Engineering (RE)*, 2003.

[21] P. Inverardi and et al. Synthesis of correct and distributed adaptors for component-based systems: an automatic approach. In *Int'l Conf. on Automated Software Engineering (ASE)*, pages 405–409, 2005.

[22] W. Jiao and H. Mei. Automating integration of heterogeneous cots components. In *Proc. 9th Int'l Conf. Software Reuse*, pages 29–42, 2006.

[23] W. Kongdenfha and et al. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Transaction on Services Computing*, 2(2):94–107, 2009.

[24] Y. Lee and et al. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.

[25] D. Mandelin, D. Kimelman, and D. M. Yellin. A bayesian approach to diagram matching with application to architectural models. In *Proc. ICSE*, pages 222–231, 2006.

[26] S. Nejati and et al. Matching and merging of statecharts specifications. In *Proc. ICSE*, pages 54–64, 2007.

[27] H. R. M. Nezhad, B. Benatallah, F. Casati, and F. Toumani. Web services interoperability specifications. *IEEE Computer*, 39(5):24–32, 2006.

[28] H. R. M. Nezhad and et al. Semi-automated adaptation of service interactions. In *Proc. 16th World Wide Web Conf. (WWW 2007)*, pages 993–1002, 2007.

[29] T. I. I. of Technology. *OntoBuilder 2 Tool for Automatic Schema Matching using Ontological Constructs*. http://iew3.technion.ac.il/OntoBuilder/, 2007.

[30] A. A. Patil and et al. Meteor-s web service annotation framework. In *Proc. 13th World Wide Web Conf. (WWW 2004)*, pages 553–562, 2004.

[31] S. R. Ponnekanti and A. Fox. Interoperability among independently evolving web services. In *Proc. Middleware*, pages 331–351, 2004.

[32] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[33] E. Rahm, H. H. Do, and S. Massmann. Matching large xml schemas. *SIGMOD Record*, 33(4):26–31, 2004.

[34] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1984.

[35] W. Tan, Y. Fan, and M. Zhou. A petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, 6(1):94–106, 2009.

[36] Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In *Proc. WISE*, pages 147–156, 2003.

[37] A. Wombacher and et al. Matchmaking for business processes based on choreographies. In *Proc. of EEE04*, pages 359–368, 2004.

[38] J. Wu and Z. Wu. Similarity-based web service matchmaking. In *Proc. SCC*, 2005.

[39] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM TOPLAS*, 19(2):292–333, 1997.

[40] A. M. Zaremski and J. M. Wing. Signature matching: A tool for using software libraries. *ACM TOSEM*, 4(2):146–170, 1995.

[41] A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM TOSEM*, 6(4):333–369, 1997.