HAL
open science

Measuring the Compatibility of Service Interaction Protocols

Meriem Ouederni, Gwen Salaün, Ernesto Pimentel

To cite this version:

Meriem Ouederni, Gwen Salaün, Ernesto Pimentel. Measuring the Compatibility of Service Interaction Protocols. 26th ACM Symposium on Applied Computing, ASME, Mar 2011, Taichung, Taiwan. hal-00650529

HAL Id: hal-00650529

https://inria.hal.science/hal-00650529v1

Submitted on 10 Dec 2011

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.

Measuring the Compatibility of Service Interaction Protocols

Meriem Ouederni
University of Málaga, Spain
meriem@lcc.uma.es

Gwen Salaün
Gr. INP–INRIA–LIG, France
Gwen.Salaun@inria.fr

Ernesto Pimentel
University of Málaga, Spain
ernesto@lcc.uma.es

ABSTRACT

Checking the compatibility of service interfaces allows one to avoid erroneous executions when composing the services together. This task is especially difficult when considering interaction protocols, that is messages and their application order, in service interfaces. Although service compatibility has been intensively studied, in particular for discovery purposes, most of existing works return a Boolean result. However, if two services are incompatible, these approaches do not indicate whether the services are almost compatible or totally incompatible. This information is crucial if one wants to apply adaptation techniques, for instance, to successfully compose these services in spite of existing mismatches. In this paper, we propose a generic flooding-based techniques for measuring the compatibility degree of service protocols. We illustrate our approach with two compatibility notions, namely *unspecified receptions* and *unidirectional complementarity*. Our solution is fully automated by a prototype tool we have implemented.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*

General Terms

Verification, Measurement

Keywords

Service Interfaces, Interaction Protocols, Formal Verification, Compatibility Flooding

1. INTRODUCTION

In today's Service Oriented Computing (SOC), organizations increasingly tend to fulfill complex requirements by composing existing services. These services have been independently developed in heterogeneous platforms without any knowledge of how to interact with each other. They are accessed through their interfaces which distinguish several interoperability levels (*i.e.*, signature, interaction protocol, quality of service, and semantics). A key issue in this setting is to check whether the service interfaces are compatible or not. This check guarantees the safe reuse and the successful interoperation of services. In this paper, we focus on the interaction protocol level of service interfaces. Checking the compatibility of interaction protocols is a tedious and difficult task even though this is of utmost importance to avoid run-time errors, *e.g.,* deadlock situations or unmatched messages. Most of the existing approaches (see for instance [8, 23, 3, 10, 4]) detect whether services are compatible or not by returning a "True" or "False" result. Unfortunately, a Boolean answer is not very helpful for many reasons. First, in real world case studies, there will seldom be a perfect match, and when service protocols are not compatible, it is useful to differentiate between services that are slightly incompatible and those that are totally incompatible. Furthermore, a Boolean result does not give a detailed measure of which parts of service protocols are compatible or not.

To overcome the aforementioned limits, a new solution aims at measuring the compatibility degree of service interfaces. This issue has been addressed by a few recent works, see for instance [22, 17, 12, 2]. However, the limitations of existing works are as follows:

- Most of them are based upon description models of service interfaces, *e.g.*, business process models [17], which do not consider value-passing with exchanged messages, and internal behaviours (τ transitions). Internal behaviours in interface models are very important while checking compatibility because some services can be compatible from an observable point of view, but their execution will behave erroneously due to these internal behaviours.
- Existing approaches such as [22] measure the interface compatibility using a simple (*i.e.*, not iterative) traversal of protocols and the results lack the preciseness which is essential for detecting subtle protocol mismatches.
- A unique compatibility notion is always considered to check the services, and this makes the approaches useful only for specific application areas, *e.g.*, service choreography [12] or adaptation [17].

In this paper, we propose a generic framework where the compatibility degree of service interfaces can be automatically measured according to different compatibility notions.

.

We illustrate our approach using both a bidirectional and an unidirectional compatibility notion, namely *unspecified receptions* and *unidirectional complementarity*. Additional notions can easily be added to our framework. The genericity of our framework makes it applicable to different application scenarios. We consider a formal model for describing service interfaces with interaction protocols (messages and their application order, but also value-passing and internal actions). In our approach, the compatibility is measured in two steps. The first step computes a set of static compatibility degrees where the execution order of messages is not taken into account. Then a flooding algorithm computes the detailed compatibility degrees of all state matches in the interaction protocols using the static compatibility results. To make the measurement more precise, our flooding algorithm combines a forward and backward compatibility propagation. This comparison process also returns a global compatibility degree and a list of mismatches indicating the interoperability issues. The proposed framework is fully automated by a prototype tool (called Comparator) which we have implemented and validated on many examples, which showed the high precision of our results.

Measuring the compatibility degree brings more advantages than the Boolean approaches and this opens a wide range of applications, in particular automatic service adaptation [14]. If a set of services is incompatible, the detailed measures and the mismatch list help to understand what parts of these services do not match. Thus, the mismatches can be worked out using adaptation techniques, and service composition can be achieved in spite of existing mismatches. Also, the computation of a global and unique compatibility degree from the detailed measures helps in ranking and selecting some services from many possible candidates.

The remainder of this paper is structured as follows. Section 2 describes our model of services. Section 3 introduces the compatibility notions we use in this paper for illustration purposes. In Section 4, we present our solution for measuring the service compatibility. Section 5 introduces our prototype tool and some experimental results. Section 6 states a brief comparison with related approaches. Finally, Section 7 draws some conclusions. All the formal definitions are given in a companion technical report [18].

2. SERVICE MODEL

We assume service interfaces are described using their interaction protocols represented by *Symbolic Transition Systems* (STSs) which are Labelled Transition Systems extended with value-passing (parameters coming with messages). Our STS is a variant of STG (Symbolic Transition Graph) presented in [11], where guards are abstracted here as transitions labelled with τ actions. A STS is a tuple (A, S, I, F, T) where: A is an alphabet which corresponds to the set of labels associated to transitions, S is a set of states, $I \in S$ is the initial state, $F \subseteq S$ is a nonempty set of final states, and $T \subseteq S \backslash F \times A \times S$ is the transition relation. In our model, a *label* is either the (internal) τ action or a tuple (m, d, pl) where m is the message name, d stands for the communication direction (either an emission ! or a reception ?), and pl is either a list of typed data terms if the label corresponds to an emission (output action), or a list of typed variables if the label is a reception (input action).[1] Communication between services relies on a synchronous and binary communication model. The operational semantics of this model is given in [10]. STSs can also be easily derived from higher-level description languages such as Abstract BPEL; see for instance [20, 5] where such abstractions were used for verification, composition or adaptation of Web services.

[1]The message names and parameter types respect the service signature.

3. PROTOCOL COMPATIBILITY

Compatibility checking verifies the successful interaction between services *w.r.t.* a criterion set on their observable actions. This criterion is referred to as a compatibility notion. In this paper, we distinguish two classes of notions depending on the direction of the compatibility checking. We refer to these classes as bidirectional and unidirectional checking. We particularly illustrate our approach with a bidirectional compatibility notion, namely *unspecified receptions* (*UR* for short), and with an unidirectional notion, namely *unidirectional complementarity* (*UC* for short).

3.1 Preliminaries

This section introduces some basic concepts needed to define the *UR* and *UC* compatibility notions. We describe a transition using a tuple (s, l, s') such that s and s' denote the source and target states, respectively, and l stands for its label. We suppose that for all transitions (s, τ, s'), $s \neq s'$. Given two services described using STSs $STS_{i \in \{1,2\}} = (A_i, S_i, I_i, F_i, T_i)$, we define a global state as a pair of states $(s_1, s_2) \in S_1 \times S_2$. For the sake of comprehension, we have chosen to present several simple examples instead of a single running example. However, we have applied our approach to many real-world case studies. Some of them are mentioned in Section 5 and others are available online [1].

Parameter Compatibility. The usual meaning of parameter compatibility requires that the parameter list expected to be received perfectly matches (same types in the same order) the parameter list coming with the sent message.

Label Compatibility. Two labels are considered compatible if they have opposite directions, same names, and compatible parameters.

Reachable States. These are the global states that the interacting protocols can access, in zero or more steps, from a current global state. Protocols can move into reachable states through synchronisations on compatible labels or independent evolutions, *i.e.*, τ transitions.

Example 1. *Figure 1 shows an example of two service protocols, which enable a database to be updated once a user account is created. As we can observe, the protocols can initially transit from (s_1, c_1) to state (s_2, c_2) through the compatible labels* register?id:int *and* register!id:int. *However, both protocols cannot synchronise on the update message because* update? *is not compatible with any label in c_1. Applying the same reasoning on (s_2, c_2), the set of global states reachable from the initial one is $\{(s_2, c_2), (s_1, c_3), (s_3, c_4)\}$.*

Deadlock-freedom. An important property required for checking the successful termination of the system is deadlock-freedom. Two protocols are considered deadlock-free at a given global state (s_1, s_2) if and only if either this state is final, *i.e.*, $(s_1, s_2) \in F_1 \times F_2$, or these protocols are deadlock-free in each global state reachable from the current one.

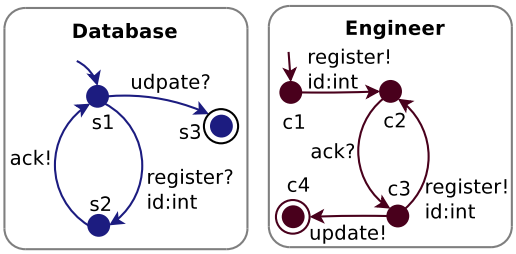

Figure 1: Database Handling System (I).

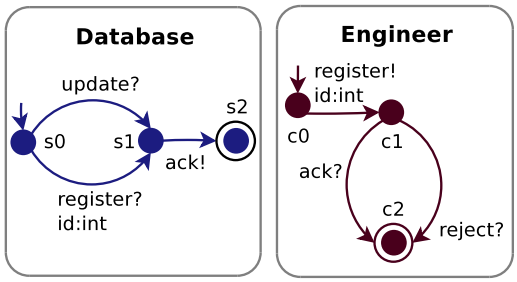

Figure 2: Database Handling System (II).

State Compatibility. Given a global state (s_1, s_2), state compatibility consists in checking whether the message l_1 sent (received, respectively) by protocol 1 at state s_1 will be eventually received (sent, respectively) by protocol 2 at state s_2, such that both protocols evolve into a compatible global state, and vice-versa. If protocol 2 is not able to interact with protocol 1's action, then both protocols must be able to reach a global state (s_1, s_2') in which this action will be satisfied, *i.e.*, $\exists(s_2', l_2, s_2'') \in T_2$ such that l_1 and l_2 are compatible, and vice-versa. The protocols must also be compatible in (s_1, s_2') and (s_1', s_2''). Since services can evolve independently through some τ transitions, the behavioural compatibility requires that *each* internal evolution must lead both services into compatible states [6, 8]. Hence, every time a τ transition is traversed in one protocol, the compatibility has to be checked again on the target state.

Example 2. *To illustrate the concept of state compatibility, we show how the compatibility can be verified at the global state (s_1, c_1) of protocols* Database *and* Engineer *in Figure 1. Although the label* register?id:int *at state s_1 can match the label* register!id:int *at state c_1, this is not the case of the label* update? *at state s_1 because it does not match any label at state c_1. However, both STSs are able to reach the global state (s_1, c_3) in which the label* update? *can be matched. Moreover,* Database *and* Engineer *are compatible in (s_1, c_3) and also in (s_3, c_4). As we can observe, every synchronisation leads both STSs into compatible global states, and the state compatibility is therefore satisfied in (s_1, c_1).*

3.2 Notions of Protocol Compatibility

Unspecified Receptions. This notion is inspired from [23] and requires that two services are compatible (i) if they are deadlock-free at their initial global state, and (ii) if one service sends a message at a reachable state, then its partner eventually receives that emission such that both services evolve into a compatible global state. The second condition is checked using the verification of state compatibility over the emission transitions. In real-life cases, one service must receive all requests from its partner, but can also be ready to accept other receptions, since the service could interoperate with other partners. Hence, there might be additional unmatched receptions in reachable states, possibly followed by unmatched emissions. These emissions do not give rise to an incompatibility issue as long as their source states are unreachable when protocols interact with each other.

Example 3. *Let us illustrate the verification of the UR compatibility on the* Engineer *and* Database *protocols in Figure 2. At the initial global state (s_0, c_0), there is a unique emission,* register!id:int*, which perfectly matches with the receptions* register?id:int*. There is also an unmatched reception* update? *at state s_0 but this does not raise an incompatibility issue according to the above definition. At the global state (s_1, c_1), the unique emission* ack! *perfectly matches with* ack?*, and here again there is an additional reception* reject?*. Moreover, these protocols do not deadlock. As a result, they are compatible w.r.t. the UR notion.*

Unidirectional Complementarity. Two services are compatible *w.r.t.* the *UC* notion if there is one service (*complementer*) which must eventually receive (send, respectively) all messages that its partner (*complemented*) expects to send (receive, respectively) at all global reachable states. In addition, both services must be deadlock-free in all reachable global states. Hence, the *complementer* service may send and receive more messages than the *complemented* service.[2] This asymmetric notion is useful for checking the successful communication in the client/server model where a server can interact with clients having different behaviours. In this setting, each client behaviour must be satisfied by the server.

Example 4. *Figure 3 consists of two protocols: the* Subscriber *(complemented) first asks for a conference registration and waits for an acknowledgment. The conference server* ConfServer *(complementer) can receive a request for either a registration or an update. Then, the server sends back to the subscriber an acknowledgement followed by a confirmation email, or terminates if this confirmation has not to be sent (described with a τ transition). We notice that the* ConfServer *complements the* Subscriber *because every time the* Subscriber *wants to transit into another state the* ConfServer *enables that transition. Moreover, both protocols are free of deadlocks. Although there is an unmatched emission* email! *in the reachable global state (s_2, c_3), the protocols remain compatible w.r.t. the UC notion, because this emission is in the complementer protocol. However, they are not compatible w.r.t. the UR notion because of this reachable but unmatched emission.*

4. MEASURING COMPATIBILITY

This section presents our techniques for measuring the compatibility of two service protocols. All the compatibility measures we present below belong to [0..1] where 1 means a perfect compatibility. The approach illustrated in Figure 4 consists first in computing a set of static compatibility measures (Section 4.1). In a second step, these static measures are used for computing the behavioural compatibility degree

[2]Our definition is different to simulation or preorder relations [7] since we compare protocols with opposite directions.

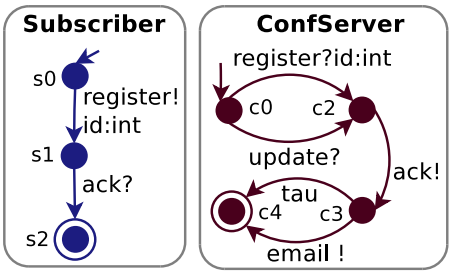

Figure 3: Conference Registration System.

for all global states in $S_1 \times S_2$ (Section 4.2). Lastly, the result is analysed and a global compatibility degree is returned (Section 4.3).

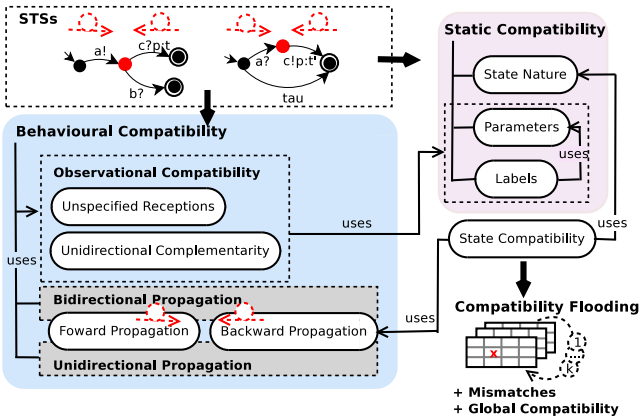

Figure 4: Compatibility Measuring Process.

4.1 Static Compatibility

We use three auxiliary static compatibility measures, namely state nature, labels, and exchanged parameters.

State Nature. The comparison of state nature assigns 1 to each pair of states which have the same nature, *i.e.*, both states are initial, final or none of them. Otherwise, the measure is 0.

Parameters. The compatibility degree of two parameter lists pl_1 and pl_2 depends on three auxiliary measures, namely: (i) the compatibility of parameter number comparing the list sizes; (ii) the compatibility of parameter order measuring the number of types which do not appear in the same order, and (iii) the compatibility of parameter type using the set of unshared types in both lists. These measures must be set to 1 if these lists are empty. Otherwise, each measure is obtained as follows: First, we compute the score of the respective mismatch, *i.e.,* different lengths of parameter lists, unordered and/or unshared types in both parameter lists. Then, we normalise the score by the maximal value that can be achieved. Finally, we decrease the mismatch score from the perfect compatibility degree (1) to obtain the auxiliary measure. The parameter compatibility degree is computed as the average of the auxiliary measures.

EXAMPLE 5. *Let us consider two parameter lists* $pl_1 = (usr{:}str, pwd{:}int)$ *and* $pl_2 = (log{:}str, sig{:}float, pwd{:}int)$. *We show below the computation of the aforementioned measures:*

- *The number compatibility is equal to* $1 - \frac{3-2}{3} = 0.66$. *In the worst case, a non-empty parameter list can be compared with an empty one. Therefore, the denominator must be set as the maximal size among those of* pl_1 *and* pl_2.
- *The order compatibility is equal to* $1 - \frac{1}{2} = 0.5$ *since* pl_1 *and* pl_2 *have one unordered type among two types existing in both lists.*
- *The type compatibility is equal to* $1 - \frac{1}{5} = 0.8$ *because* pl_2 *does not share the type* $float$ *with* pl_1. *The number of unshared types is normalised with the sum of* pl_1 *and* pl_2 *sizes because in the worst case both lists could have types totally different.*
- *As a consequence, the parameter compatibility is equal to* $\frac{0.66+0.5+0.8}{3} = 0.65$.

Labels. Protocol synchronisation requires that compatible labels must have opposite directions. Therefore, given a pair $(l_1, l_2) \in A_1 \times A_2$, the label compatibility is measured as 0 if these labels have same directions. Otherwise, the computation of this measure uses the semantic distance between message names and the parameter compatibility degree presented above. Here, message names are compared using the Wordnet similarity package. Note that message names and parameters can be compared using other techniques such as the N-gram algorithm [13]. It is also possible to compare the semantics of parameter names and/or types using the Wordnet similarity package.

4.2 Behavioural Compatibility

We consider a flooding algorithm which performs an iterative measuring of behavioural compatibility for every global state in $S_1 \times S_2$. This algorithm incrementally propagates the compatibility between neighbouring states using backward and forward processing. The compatibility propagation is based on the intuition that two states are compatible if their backward and forward neighbouring states are compatible. Note that the backward and forward neighbours of the global state (s'_1, s'_2) in the transition relations $T_1 = \{(s_1, l_1, s'_1), (s'_1, l'_1, s''_1)\}$ and $T_2 = \{(s_2, l_2, s'_2), (s'_2, l'_2, s''_2)\}$ are the states (s_1, s_2) and (s''_1, s''_2), respectively. The flooding algorithm returns a matrix denoted $COMP^k_{CN,D}$ where each entry $COMP^k_{CN,D}[s_1, s_2]$ stands for the compatibility measure of global state (s_1, s_2) at the kth iteration. The parameter CN refers to the considered compatibility notion which must be checked according to D that is either an unidirectional ($\rightarrow$) or a bidirectional ($\leftrightarrow$) protocol analysis. $COMP^0_{CN,D}$ represents the initial compatibility matrix where all states are supposed to be perfectly compatible, *i.e.*, $\forall (s_1, s_2) \in S_1 \times S_2,\ COMP^0_{CN,D}[s_1, s_2] = 1$. Then, in order to compute $COMP^k_{CN,D}[s_1, s_2]$, we need two functions, namely $obs\text{-}comp^k_{CN,D}$ and $state\text{-}comp^k_{CN,D}$ that we detail in the following. The first function computes the compatibility of outgoing (incoming, respectively) observable transitions being given a compatibility notion CN. We refer to this measure as observational compatibility. The second function propagates the compatibility from the forward and backward (denoted fw and bw for short, and illustrated in Figure 4 with red dashed arrows) neighbouring states to (s_1, s_2) taking into account τ transitions. Thus, the computation of $state\text{-}comp^k_{CN,D}$ combines two auxiliary functions,

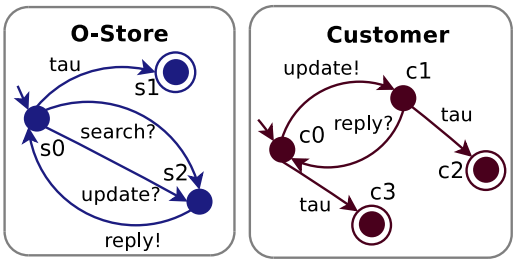

Figure 5: Online Store (I).

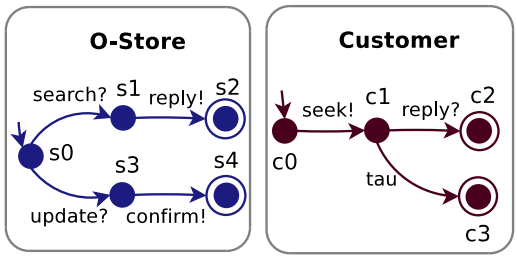

Figure 6: Online Store (II).

namely $fw\text{-}propag^k_{CN,D}$ and $bw\text{-}propag^k_{CN,D}$. In this paper, we only present the forward compatibility for lack of space, the backward compatibility can be computed in a similar way based upon incoming rather than outgoing transitions. In the following, we start by introducing the computation of observational compatibility *w.r.t.* to *UR* and *UC* notions presented in Section 3.2.

Unspecified Receptions. For all global states (s_1, s_2): (i) $obs\text{-}comp^k_{UR,\leftrightarrow}$ returns 1 if and only if every outgoing emission at state s_1 (and s_2) perfectly matches an outgoing reception at state s_2 (and s_1) and all synchronisations on those emissions lead to compatible states; (ii) $obs\text{-}comp^k_{UR,\leftrightarrow}$ returns 0 if there is a deadlock; (iii) otherwise, $obs\text{-}comp^k_{UR,\leftrightarrow}$ measures the best compatibility of every outgoing emission at s_1 with the outgoing receptions at s_2, leading to the neighbouring states which have the highest compatibility degree, and vice-versa.

EXAMPLE 6. *Let us consider the global state* (s_0, c_0) *in Figure 5. Here, there is a unique emission* update! *at* c_0 *which perfectly matches with the reception* update? *at* s_0, $lab\text{-}comp(update!, update?) = 1$. *The synchronisation on these compatible labels leads to the target global state* (s_2, c_1) *where* $COMP^0_{UR,\leftrightarrow}[s_2, c_1] = 1$. *Thus, at the first iteration:* $obs\text{-}comp^1_{UR,\leftrightarrow}((s_0, c_0))$
$= lab\text{-}comp(update!, update?) * COMP^0_{UR,\leftrightarrow}[s_2, c_1] = 1$.

Unidirectional Complementarity. We assume that one state s_{er} (in the *complementer* protocol) perfectly complements the state s_{ed} (in the *complemented* protocol), *i.e.*, $obs\text{-}comp^k_{UC,\rightarrow}((s_{er}, s_{ed})) = 1$, if there is a subset of outgoing observable transitions at s_{er} such that their respective labels are perfectly compatible with those of transitions at s_{ed}. In addition, these transitions must lead to compatible states. If there is a deadlock, this function returns 0. Otherwise, $obs\text{-}comp^k_{UC,\rightarrow}((s_{er}, s_{ed}))$ measures the best compatibility of every transition label going out from s_{ed} with those of transitions going out from s_{er}, leading to the neighbouring states which have the highest compatibility degree.

EXAMPLE 7. *Let us consider the global state* (s_0, c_0) *in Figure 6. We want to check whether the* O-Store *protocol complements the* Customer *protocol. Initially, there is only one observable message* seek! *in the* Customer *protocol which perfectly matches with message* search? *in the* O-Store *protocol (message names are synonyms in the Wordnet similarity package),* $lab\text{-}comp(seek!, search?) = 1$. *In addition, the protocols can reach* (s_1, c_1) *through the synchronisation on these compatible labels such that* $COMP^0_{UC,\rightarrow}[s_1, c_1] = 1$. *Therefore, at the first iteration,* $obs\text{-}comp^1_{UC,\rightarrow}((s_0, c_0)) = lab\text{-}comp(seek!, search?) * COMP^0_{UC,\rightarrow}[s_1, c_1] = 1$.

As far as τ transitions are concerned, we define the function $fw\text{-}propag^k_{CN,D}$, $D \in \{\leftrightarrow, \rightarrow\}$, which handles these internal behaviours based upon either a bidirectional or unidirectional compatibility propagation:

Bidirectional Propagation. The protocol compatibility is analysed from the point of view of both services. The function $fw\text{-}propag^k_{CN,\leftrightarrow}((s_1, s_2))$ propagates to (s_1, s_2) the compatibility degrees obtained for the forward neighbours of state s_1 with those of state s_2, and vice-versa. For each τ transition, $fw\text{-}propag^k_{CN,\leftrightarrow}$ must be checked on the target state. Observable transitions going out from (s_1, s_2) are compared using $obs\text{-}comp^k_{CN,\leftrightarrow}((s_1, s_2))$.

EXAMPLE 8. *Let us consider again the global state* (s_0, c_0) *in Figure 5 and the UR notion. We show below the computation of* $fw\text{-}propag^1_{UR,\leftrightarrow}$ *at the initial global state, and which results in the average of the auxiliary values computed from each protocol point of view:*

$$fw\text{-}propag^1_{UR,\leftrightarrow}((s_0, c_0)) = \frac{1}{2} * \left(\frac{fw\text{-}propag^1_{UR,\leftrightarrow}((s_1,c_0)) + obs\text{-}comp^1_{UR,\leftrightarrow}((s_0,c_0))}{2} + \frac{fw\text{-}propag^1_{UR,\leftrightarrow}((s_0,c_3)) + obs\text{-}comp^1_{UR,\leftrightarrow}((s_0,c_0))}{2} \right)$$

where:

- $fw\text{-}propag^1_{UR,\leftrightarrow}((s_1, c_0)) = obs\text{-}comp^1_{UR,\leftrightarrow}((s_1, c_0)) = 0$ *due to the deadlock that can occur at the global state* (s_1, c_0).
- $fw\text{-}propag^1_{UR,\leftrightarrow}((s_0, c_3)) = obs\text{-}comp^1_{UR,\leftrightarrow}((s_0, c_3)) = 0$ *because both protocols can also deadlock at the global state* (s_0, c_3).
- $obs\text{-}comp^1_{UR,\leftrightarrow}((s_0, c_0)) = lab\text{-}comp(update?, update!) * COMP^0_{UR,\leftrightarrow}[s_2, c_1] = 1$.

As a consequence, $fw\text{-}propag^1_{UR,\leftrightarrow}((s_0, c_0)) = \frac{1}{2}$.

Unidirectional Propagation. We assume that the function $fw\text{-}propag^k_{CN,\rightarrow}((s_1, s_2))$ is computed from STS_2's point of view such that the service compatibility is governed by STS_2's requirements. First, if no τ transition exists at (s_1, s_2), $fw\text{-}propag^k_{CN,\rightarrow}((s_1, s_2)) = obs\text{-}comp^k_{CN,D}((s_1, s_2))$. If there exist τ transitions at s_1, this state is considered compatible with s_2 if $fw\text{-}propag^k_{CN,\rightarrow}((s'_1, s_2)) = 1$ for every (s_1, τ, s'_1). This check ensures that each time STS_1 traverses an internal transition, this protocol must be able to fulfill STS_2's requirements at the target state. If the last condition does not hold, we need to compute $fw\text{-}propag^k_{CN,\rightarrow}((s'_1, s_2))$ for every (s_1, τ, s'_1), and also the compatibility of observable transitions going out from the global state (s_1, s_2) obtained using $obs\text{-}comp^k_{CN,D}((s_1, s_2))$. Otherwise, if no (s_1, τ, s'_1) exists, we compute $fw\text{-}propag^k_{CN,\rightarrow}((s_1, s'_2))$ for every (s_2, τ, s'_2), and also $obs\text{-}comp^k_{CN,D}((s_1, s_2))$.

	s_0	s_1	s_2	s_3	$s4$
c_0	**0.78**	0.01	0.01	0.01	0.01
c_1	0.01	0.68	0.01	0.35	0.01
c_2	0.01	0.01	0.90	0.01	0.67
c_3	0.01	0.45	0.76	0.35	0.76

Table 1: The Compatibility Matrix $COMP^7_{UC,\rightarrow}$.

EXAMPLE 9. *Let us show the computation of the function $fw\text{-}propag^1_{UC,\rightarrow}$ at states (s_0, c_0) and (s_1, c_1) in Figure 6. First, $fw\text{-}propag^1_{UC,\rightarrow}((s_0, c_0)) = obs\text{-}comp^1_{UC,\rightarrow}((s_0, c_0)) = 1$ because no τ transition exists. Since there is one τ transition at c_1:*
$fw\text{-}propag^1_{UC,\rightarrow}((s_1, c_1)) =$

$$\frac{fw\text{-}propag^1_{UC,\rightarrow}((s_1, c_3)) + obs\text{-}comp^1_{UC,\rightarrow}((s_1, c_1))}{2}$$

where:

- $fw\text{-}propag^1_{UC,\rightarrow}((s_1, c_3)) = obs\text{-}comp^1_{UC,\rightarrow}((s_1, c_3)) = 0$ *due to the deadlock at state* (s_1, c_3).
- $obs\text{-}comp^1_{UC,\rightarrow}((s_1, c_1)) = lab\text{-}comp(reply!, reply?) * COMP^0_{UC,\leftrightarrow}[s_2, c_2] = 1$.

Hence, $fw\text{-}propag^1_{UC,\rightarrow}((s_1, c_1)) = \frac{1}{2}$.

State Compatibility. The function $state\text{-}comp^k_{CN,D}((s_1, s_2))$ computes the weighted average of three measures: the forward and backward compatibilities, and the value returned by the function comparing state natures.

Compatibility Flooding. Finally, $COMP^k_{CN,D}[s_1, s_2]$ is computed as the average of its previous value at the $k-1^{th}$ iteration and the current state compatibility degree. Our iterative process terminates when the Euclidean difference $\|COMP^k_{CN,D} - COMP^{k-1}_{CN,D}\|$ converges.

EXAMPLE 10. *Table 1 shows the matrix computed for the example depicted in Figure 6 according to the UC notion. This matrix was obtained after 7 iterations. Let us comment on the compatibility of states c_0 and s_0. The measure is quite high because both states are initial and the emission* seek! *at c_0 perfectly matches the reception* search? *at s_0. However, the compatibility degree is less than 1 due to the backward propagation of the deadlock from the global state (s_1, c_3) to (s_1, c_1), and then from (s_1, c_1) to (s_0, c_0).*

Mismatch List. Our compatibility measure comes with a list of mismatches which identifies the incompatibility sources, *e.g.,* unmatched message names or unshared parameter types. For instance, the states s_0 and c_1 in Figure 6 present several mismatches, *e.g.*, the first state is initial while the second is not, and their outgoing transition labels have the same directions.

Extensibility. Our approach is generic and can be easily extended to integrate other compatibility notions. Adding a compatibility notion *CN* only requires definition of a new function $obs\text{-}comp^k_{CN,D}$.

4.3 Analysis of Compatibility Measures

In this section, we propose some techniques for automatically analysing the measures obtained from the compatibility matrix. We first present how the Boolean compatibility can be computed from the matrix. In the case of incompatible services, we propose some techniques for computing a global compatibility measure.

Compatible Protocols. Our flooding algorithm ensures that every time a mismatch is detected in a reachable global state, its effect will be propagated to the initial states. Hence, the forward and backward compatibility propagation between neighbouring states implies that protocols are compatible if and only if their initial states are also compatible. Such information is useful for automatically discovering available services that can be composed without using any adaptor service for compensating mismatches.

Global Protocol Compatibility. The global compatibility measure helps to differentiate between services that are slightly incompatible and those which are totally incompatible. This is useful to perform a first service ranking and selection step to find some candidates among a large number of services. Seeking services with high global compatibility degree enables simplification of further processing to compensate existing mismatches, *e.g.*, using service adaptation [14].

The global compatibility can be computed differently depending on the user's preferences. One solution consists in computing the average of the maximal compatibility degrees obtained for all states. An alternative is to compute the global compatibility degree as the weighted average of all behavioural compatibility degrees that are higher than a threshold t. The weight is the rate of states having a compatibility degree higher than t, among all states compared in one service, with the states in the partner service. So far, our approach supports the second solution. The computation algorithm is given in [18].

5. PROTOTYPE TOOL

Our approach for measuring the compatibility degree of service protocols has been fully implemented in a prototype tool called Comparator [1]. The framework's architecture is given in Figure 7. The Comparator prototype tool has been implemented in Python 2.6 using Eclipse 3.5.1 as the programming IDE. The tool accepts as input two XML files corresponding to the service interfaces and an initial configuration, *i.e.*, the compatibility notion, the checking direction, and a threshold t. As a result, Comparator returns the compatibility matrix, the mismatch list, and the global compatibility degree which indicates how compatible both services are. The implementation of our proposal is highly modular (as shown in Figure 7) which facilitates its extension with new compatibility notions, as well as other strategies for comparing message names and parameters.

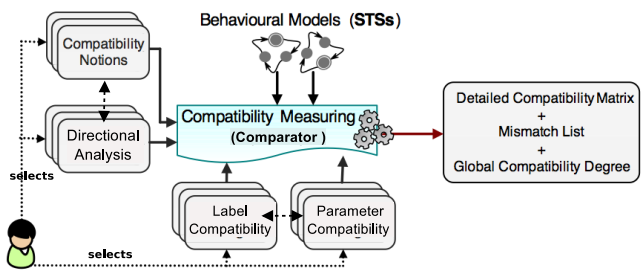

Figure 7: Comparator Architecture.

Experimental Results. So far, we have validated our prototype tool on more than 110 real-world examples, *e.g.*, a car rental, a travel booking system, a hard disk manager, a medical management system, an on-line email service. It is worth noticing that Comparator computes the compatibility degree of quite large systems (*e.g.*, services with hundreds of states and transitions) in a reasonable time (a few minutes) on a Mac OS machine running on a 2.53 GHz Intel dual core processor with 4 GB of RAM. Several case studies illustrating the computation of our compatibility measure are presented in [18, 1].

Evaluation. As regards accuracy, we reuse the well-known precision and recall metrics to estimate how much the measure automatically computed meets the expected result. Precision measures the matching quality (number of false positive matching) and is defined as the ratio of the number of correct state matching found to the total of state matching found. Recall is the coverage of the state matching results and is defined as the ratio of the number of correct state matching found to the total of all correct state matching in the two protocols. An effective measure must produce high precision and recall values. We have studied the precision and recall for the examples of our database. We assume (s_i, s_j) is a correct match if the state $s_i \in S_i$ has the highest compatibility degree with $s_j \in S_j$ among those in S_j. Our measuring process yields a precision and recall of 100% for compatible protocols. Our empirical analysis has also shown the good quality of our approach for comparing incompatible protocols. For instance, the study of the car rental system [1] – which provides a service for car rental and an example of user requirements – produces a precision and recall equal to 85% and 95%, respectively. We applied the same evaluation to a flight advice system [1] which helps travelers to find flight information. This yields a precision and a recall equal to 91% and 100%, respectively. All the other examples of our database returned high values (more than 90%) for these two metrics.

6. RELATED WORK

Existing quantitative techniques devoted to behavioural analysis focus on two closely-related research problems. The first one is called substitutability checking and aims at finding correspondences between similar services. The second one is referred to as compatibility checking and verifies whether interacting services fulfill each other's requirements. Let us focus on three kinds of approaches existing for these issues.

Simple Protocol Traversal. [21] measures the similarity of Labelled Transition Systems (LTSs) *w.r.t.* a simulation and a bisimulation notion inspired from the equivalence relations. The measuring techniques use weighted quantitative functions which consist in a simple (not iterative), forward, and parallel traversal of two LTSs. This work does not return the global similarity degree and the differences which distinguish one service from another. In [22], two services described using π-calculus are considered compatible if there is always at least one transition sequence between them until reaching final states, but this does not guarantee the deadlock-freedom. A simple and parallel traversal of protocols computes the compatibility degree as the average of the number of successful transition sequences. This work does not measure the detailed compatibility of different protocol states, and there is no mismatch detection.

Edit Distance. In [12, 2], the authors calculate the minimal edit distance between two versions of one service interface. [12] extends the simulation algorithm given in [21] in order to correct deadlocking choreographies. In particular, it detects the modifications needed to achieve service simulation and make the choreography free of deadlocks. In contrast, [2] measures the state simulation based on the analysis of outgoing transition labels without any semantic comparison of these label names. This measuring technique does not consider the similarity of neighbouring states, therefore the main advantage of a propagation-based approach is missing. This approach computes a global similarity measure.

Similarity Flooding. In [15, 16], the authors rely on a similarity flooding algorithm for computing the matrix of correspondences between models. [15] considers a forward and backward similarity propagation to compare data structures described with directed labelled graphs. However, the tool does not enable a fully automated matching because the user should manually adjust some matches. The *match* operator introduced in [16] measures the similarity between different versions of software units described using Statecharts. The similarity measuring combines a set of static and behavioural matchings. The behavioural matching is computed using a flooding algorithm and relies on the bisimulation notion presented in [21]. In this work, the behavioural similarity is computed as the maximum of forward and backward behavioural matching. By doing so, it is not possible to detect the Boolean similarity from the initial states. More recently, [17] propose a semi-automated approach for checking the matching of messages in two business process models such that the computed values can be updated depending on the user feedback. The authors combine a depth and flooding-based interface matching for measuring the behavioural compatibility of two interacting protocols. This work aims at detecting the message merge/split mismatch in order to help the automatic specification of adaptation contacts. A detailed discussion on the comparison of business process models is presented in [19]. Our approach is different since we focus on measuring the compatibility of process-oriented models which are understood as refinements of business process models [9]. We compare in Table 2[3] our proposal with the most related works.

7. CONCLUSION

To the best of our knowledge, we are the first to suggest a generic framework supporting different notions for measuring the compatibility degree of service interfaces. Our measuring method takes into account value-passing and internal behaviours. Considering both the forward and backward compatibility propagation makes our flooding algorithm more precise, and also enables us to detect Boolean compatibility. In addition to the matrix computation and the global measure of compatibility, a list of mismatches is returned. Our proposal is fully supported by the Comparator tool which has been validated on many examples. Our compatibility degree results have some straightforward applications for service selection and adaptation [14, 23]. Our tool has already been integrated into an environment for the interactive specification of adaptation contracts [5]. Our

[3]BIS, SIM, OP, DF, UR, UC, WK, and ST are used as abbreviations of bisimulation, simulation, one path, deadlock-freedom, unspecified receptions, unidirectional complementarity, weak, and strong, respectively.

		[16]	[21]	[12]	[2]	[22]	[17]	Our approach
Model	Messages and protocols	√	√	√	√	√	√	√
	Value-passing	×	×	×	×	×	×	√
	Internal actions	×	×	×	×	√	×	√
	Description language	Statechart	LTS	Finite Automaton	FSM	π-calculus	FSM	STS
Analysis	Issue	Similarity	Similarity	Similarity	Similarity	Compatibility	Compatibility	Compatibility
	Notion(s)	BIS	(WK/ST) SIM/BIS	WK SIM	SIM	OP	DF	UR/UC/...
Computation	Message semantics	√	×	×	×	×	√	√
	Processing	Iterative	Simple	Simple	Simple	Simple	Iterative	Iterative
	Technique	Flooding	Parallel traversal	Edit distance	Edit distance	Parallel traversal	Flooding	Flooding
	Detailed measures	√	√	√	√	×	√	√
	Mismatch detection	×	×	√	√	×	√	√
	Global measure	×	×	×	√	√	×	√
Tool support		√	√	√	√	√	√	√

Table 2: A summary of Approaches Based on Quantitative Behavioural Analysis.

main perspective is to apply our compatibility measuring approach for the automatic generation of adaptor protocols.

Acknowledgements. This work has been partially supported by the project TIN2008-05932 funded by the Spanish Ministry of Innovation and Science and FEDER, and by the project P07-TIC03131, funded by the Andalusian government. We are also grateful to Christine McKinty, who proofread the final version of this paper.

8. REFERENCES

[1] Comparator: A Tool for Measuring the Compatibility Degree of Service Protocols. Available on Meriem Ouederni's Web Page.

[2] A. Aït-Bachir. Measuring Similarity of Service Interfaces. In *Proc. of the PhD Symposium at ICSOC'08*, volume 421 of *CEUR Workshop Proceedings*, 2008.

[3] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In *Proc. of TES'04*, volume 3324 of *LNCS*, pages 15–28. Springer, 2004.

[4] M. Bravetti and G. Zavattaro. Contract-Based Discovery and Composition of Web Services. In *SFM'09*, volume 5569 of *LNCS*, pages 261–295. Springer, 2009.

[5] J. Cámara, G. Salaün, C. Canal, and M. Ouederni. Interactive Specification and Verification of Behavioural Adaptation Contracts. In *Proc. of QSIC'09*, pages 65–75. IEEE Computer Society, 2009.

[6] C. Canal, E. Pimentel, and J. M. Troya. Compatibility and Inheritance in Software Architectures. *Sci. Comput. Program.*, 41(2):105–138, 2001.

[7] R. Cleaveland and O. Sokolsky. Equivalence and Preorder Checking for Finite-State Systems. *Handbook of Process Algebra*, pages 391–424, 2001.

[8] L. de Alfaro and T. Henzinger. Interface Automata. In *Proc. of ESEC/FSE'01*, pages 109–120. ACM Press, 2001.

[9] R. M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *Proc. of BPM'09*, volume 5701 of *LNCS*, pages 48–63. Springer, 2009.

[10] F. Durán, M. Ouederni, and G. Salaün. Checking Protocol Compatibility using Maude. In *Proc. of FOCLASA'09*, volume 255, pages 65–81. ENTCS, 2009.

[11] M. Hennessy and H. Lin. Symbolic Bisimulations. *TCS*, 138(2):353–389, 1995.

[12] N. Lohmann. Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In *Proc. of BPM'08*, volume 5240 of *LNCS*, pages 132–147. Springer, 2008.

[13] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[14] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. In *Proc. of ICSOC'08*, volume 5364 of *LNCS*, pages 84–99. Springer, 2008.

[15] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proc. of ICDE'02*, pages 117–128. IEEE Computer Society, 2002.

[16] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. Matching and Merging of Statecharts Specifications. In *Proc. of ICSE'07*, pages 54–64. ACM Press, 2007.

[17] H. R. M. Nezhad, G. Y. Xu, and B. Benatallah. Protocol-aware Matching of Web Service Interfaces for Adapter Development. In *Proc. of WWW'10*, pages 731–740. ACM, 2010.

[18] M. Ouederni, G. Salaün, and E. Pimentel. Measuring the Compatibility of Service Interaction Protocols. Technical Report ITI 4-10, Dept. of LCC, University of Málaga, 2010.

[19] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling. From Business Process Models to Process-Oriented Software systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1), 2009.

[20] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. *IJBPIM*, 1(2):116–128, 2006.

[21] O. Sokolsky, S. Kannan, and I. Lee. Simulation-Based Graph Similarity. In *Proc. of TACAS'06*, volume 3920 of *LNCS*, pages 426–440. Springer, 2006.

[22] Z. Wu, S. Deng, Y. Li, and J. Wu. Computing Compatibility in Dynamic Service Composition. *Knowledge and Information Systems*, 19(1):107–129, 2009.

[23] D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.