Compatibility Flooding: Measuring Interaction of Behavioural Models

Meriem OUEDERNI
INP Toulouse, IRIT
Toulouse
France
meriem.ouederni@irit.fr

Uli FAHRENBERG
École polytechnique
Palaiseau
France
uli@lix.polytechnique.fr

AXEL LEGAY
INRIA Rennes
Rennes
France
axel.legay@inria.fr

Gwen SALAUN
Inria Grenoble LIG Grenoble
Grenoble
France
Gwen.Salaun@inria.fr

ABSTRACT

In this paper, we prove the convergency of our compatibility flooding algorithm which measures the compatibility degree of service interfaces. Our method is generic and fully automated by a prototype tool.

Keywords

Service Interfaces, Interaction Protocols, Formal Verification, Compatibility Flooding

Software systems are mostly built using existing services which interact with each other via message exchange to fulfil a common goal. Services are considered as black boxes accessed through their public interfaces which present four interoperability levels [2], *i.e.*, signature, interaction protocols, quality of services, and semantics. These interfaces must be compatible in order to ensure the correct composition and reuse of loosely-coupled services. This paper deals with the compatibility verification of service interfaces and focuses on the interaction protocol level. Checking the compatibility of interaction protocols is a tedious and hard task, even though it is of utmost importance to avoid run-time errors, *e.g.*, deadlock situations or unmatched messages.

Most of the existing approaches (see [7] for a detailed survey) return a "True" or "False" result to detect whether services are compatible or not. Unfortunately, for many issues a Boolean answer is not very helpful. Firstly, in real world situations, 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. Secondly, a Boolean result does not give any detailed information on which parts of service protocols are compatible or not. Thirdly, regarding the incompatible parts of protocols, such a result typically does not come with a mismatch list which enables us to understand and then resolve the incompatibility issues.

SAC'17, April 3-7, 2017, Marrakesh, Morocco

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

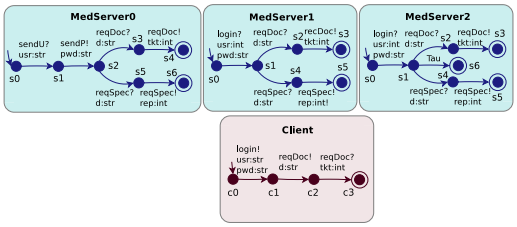


Figure 1: A Simple Medical Management System.

To overcome the aforementioned limits, a new solution aims at *quantifying* the compatibility degree of service interfaces. This issue has been addressed by a few recent works (see Section 5 for related work). However, most of them are based upon description models of service interfaces, *e.g.*, business protocols [15], which do not consider value-passing coming with exchanged messages and internal behaviours (τ transitions). Internal behaviours in interface models are very important because some services can be compatible from an observable point of view, but their execution will behave erroneously if these behaviours are not taken into account [16]. Moreover, existing approaches, such as [26], measure interface compatibility using a simple (*i.e.*, not iterative) traversal of protocols, and the results lack the preciseness essential for detecting subtle protocol mismatches. Lastly, 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 [11] or service adaptation [15].

As an example, we refer to the symbolic transition systems[1] in Fig. 1, describing an on-line medical management system which handles patient appointments within a healthcare institution, either with general practitioners or with specialist doctors. The Client first logs on to a server by sending her user name and password (login!). Then, she asks for an appointment with a general practitioner (reqDoc!) and receives an appointment identifier. We present three services for the medical server, which all seem to match the Client service, yet all fail in subtle ways: MedServer0 can only receive user name and password separately, whereas Client wants to send them together; MedServer1 has a type mismatch on the usr parameter; and with MedServer2, communication can deadlock (if it silently proceeds to state s6, for example after a timeout). Hence

[1]We shall introduce symbolic transition systems more formally in Section 1.

none of the MedServers is compatible with Client. We shall, however, later see that, using our quantitative techniques, there is a clear preference for MedServer1 in which the incompatibilities are much easier mended than in the other two.

In our previous work [17, 19], we proposed an algorithm for quantifying the compatibility degree of interacting services. We compute a numerical measure in the interval $[0..1]$, where 0 means no compatibility and 1 means perfect compatibility. We describe service interfaces using a formal model, taking into account interaction protocols, *i.e.*, messages and their application order, but also value-passing and internal actions. 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 bidirectional and unidirectional compatibility notions, namely *unspecified receptions* [27] and *unidirectional complementarity*; additional notions can easily be added to our framework. The compatibility is computed in two steps. First, we compute a set of static compatibility degrees where the execution order of messages is not taken into account. Then, we propose a new flooding algorithm to compute the compatibility degree of interaction protocols using the static compatibility results. We assume that the compatibility flooding includes backward and forward compatibility propagation among service protocols. The computation process also returns the mismatch list indicating the interoperability issues, and a global compatibility degree for two interaction protocols. Our solution is fully automated by a prototype tool Comparator [5] we have implemented.

This paper improves previous preliminary approaches [17, 19] as follows. We give the mathematical definitions of our heuristics used for computing the compatibility of two service interfaces. We prove, using Banach's fixed point theorem, that the flooding-based computation always converges. A Web application is also implemented and provided online for using our prototype tool. We present several experiments to better evaluate our prototype tool. We discuss and evaluate the new approach w.r.t. many features. Finally, we thoroughly revise related work. In particular, we classify the state-of-the-art w.r.t. several existing schools working on heuristics-based approaches.

Our quantification of protocol compatibility brings more advantages than the Boolean approaches, because it returns a detailed measure but also a list of mismatches that can be useful for many service applications, such as automatic service ranking, service discovery, composition, or adaptation. In the case of service adaptation [12] for instance, if a set of services are incompatible, the detailed measure and the mismatch list help to understand which 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. Lastly, we consider forward and backward exploration of protocols in our flooding algorithm. This enables us to detect full compatibility, *i.e.*, "true" result in the case of model checking, and incompatibility, *i.e.*, "false" result otherwise (see Section 3.3).

In the remainder of this paper, section 1 defines the used compatibility notions. Section 1 defines some compatibility notions. In Section 2, we present our approach for quantifying service compatibility. Sections gives the convergency proof and discusses the main characteristics of our algorithm. Section 4 introduces our prototype tool and some experimental results. Section 5 compares our approach with related work.

1. SERVICE COMPATIBILITY

Service compatibility is achieved if services can correctly interact with each other, *i.e.*, synchronisations over observable actions which are exchanged between services. Checking for correct service interaction needs to verify if service protocols satisfy a criterion, *i.e.*, compatibility notion. In this article, we compute the compatibility for two services and we distinguish two classes of notions, bidirectional ($\leftrightarrow$) and unidirectional ($\rightarrow$), depending on the direction of the compatibility checking. We particularly illustrate our approach with one bidirectional notion, namely *unspecified receptions* (*UR* for short), and with one unidirectional notion, namely *unidirectional complementarity* (*UC* for short).[2] In the case of unidirectional compatibility checking, one of the two services plays a particular role because its *requirements* (messages) must all be satisfied by the partner service. This class can be useful for checking, *e.g.*, a client/server model. In this setting, a server service must be able to receive and answer all requests from a client, but this server can also handle other requests from other clients.

Before defining *UR* and *UC*, we give below some preliminary concepts on which those compatibility notions rely. These concepts are detailed and formally presented in [7].

DEFINITION 1. *A Symbolic Transition System, or 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 $T \subseteq S \setminus F \times A \times S$ is the transition relation.*

A *label* is either the (internal) τ action or a tuple (m, d, pl) where m is the message name, d is 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, or a list of typed variables if the label is a reception. Here, services interact with each other based on a synchronous and binary communication model. The operational semantics of this model is formalised in [7].[3]

Static Compatibility. We focus here on two comparisons independently on the order of exchanged messages, *i.e.*, behaviour: (i) *Parameter compatibility* requires that the parameter list expected to be received matches (same types in the same order) the parameter list coming with the sent message. (ii) *Label compatibility* requires labels to have opposite directions, same names, and compatible parameters, *e.g.*, labels search?p1:t1 and search! p2:t1 are compatible where p1 and p2 are the parameter names and t1 and t2 are their respective types.

Reachable States. These are the set of all global states that interacting protocols can reach, in zero or more steps, from a current global state (s_1, s_2). Protocols can move into reachable states through either synchronisations on compatible labels or independent evolutions, *i.e.*, τ transitions.

Deadlock-Freeness. This enables us to check successful system termination, *i.e.*, the services can always interact (through synchronisations on compatible labels) with each other or evolute independently (through τ transitions) starting from a global initial state until reaching final states. All the traversed global states belong to the set of deadlock-free states (referred to as *DF*).

State Compatibility. Service interaction depends on synchronisation over compatible labels and is checked on reachable global states. For a given global state (s_1, s_2) of two protocols $STS_i = (A_i, S_i, I_i, F_i, T_i)$, this state is compatible if every message l_1 sent (received) by STS_1 at state s_1 will be received (sent, respectively)

[2]The reader can refer to [7] for more notions belonging to both classes.

[3]In the rest of the article we will describe service interfaces only with their corresponding STSs. Signatures will be left implicit, yet they can be inferred from the typing of arguments in STS labels. We suppose that there are no cycles of internal transitions, *i.e.*, no transition sequences $(s_1, \tau, s_2), \ldots, (s_n, \tau, s_1)$ because this can be reduced using τ confluence techniques.

by STS_2 at state s_2 (*i.e.*, l_1 matches with l_2 at s_2 where l_1 and l_2 are compatible labels) such that both protocols evolve into a compatible global state, and vice-versa. Otherwise, if STS_2 cannot synchronise with STS_1's label l_1 at (s_1, s_2), 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. In this case, both states (s_1, s_2') and (s_1', s_2'') must also be compatible. This protocol traversal can continue until reaching a final global state (both protocols states are final). Note that we handle τ actions similarly to branching equivalence [25].

Unspecified Receptions (*UR*). This notion is inspired from [27] and requires that two services are compatible if (i) they are deadlock-free, (ii) every send message at a reachable global state must be received by the partner peer such that this condition is checked by restriction of verification of state compatibility over uniquely the emission transitions, and (iii) for two communicating services, the condition (ii) must be true at their initial global state. 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.

Unidirectional Complementarity (*UC*). We consider that two services are compatible w.r.t. the *UC* notion if (i) they are deadlock-free and (ii) starting from the initial global state, one of them (the *complementer*) must eventually receive and send all messages that its partner (*complemented*) expects to send and receive, respectively, in the same order at all global reachable states. Hence, the *complementer* service may send and receive more messages than the *complemented* service. This asymmetric notion is useful to check the successful communication in the client/server model where a server can interact with clients with different behaviours. In this setting, each client behaviour must be satisfied (complemented) by the server.

2. QUANTIFYING COMPATIBILITY

We now show how to compute the compatibility of two service protocols as numerical measure which belongs to the interval [0..1] where 0 means total incompatibility and 1 means perfect compatibility. Our techniques rely on the compatibility definitions given in Section 1. We compute the compatibility at several levels of service interfaces such as states, labels, and parameters. We aim at using all information described in service interfaces in order to get the highest precision for the computed compatibility. The final compatibility degree of two interaction protocols is computed relying on all these sources of compatibility, and following a *divide-and-conquer* approach.

For purpose of clarity, we assume in the rest of this article that the different functions defined have access to the $STS_i = (A_i, S_i, I_i, F_i, T_i)$ even if they are not explicitly passed as input parameters. However, we make parameters explicit if they are modified. The full computation process overviewed in Fig. 2 is iterative. Each iteration consists first in computing three static compatibility measures (Section 2.1) where the order of exchanged messages is not considered. In a second step, these static measures are used for computing the behavioural compatibility degree for all global states (Section 2.2). Lastly, we compute the global compatibility degree and return all the detected mismatches (Section 3.3).

2.1 Static Compatibility

State Nature. We compare state nature using the function *nat*$((s_1, s_2))$. It returns 1 if states s_1 and s_2 have the same nature, *i.e.*, both are either initial, final or none of the two. Otherwise, *nat*$((s_1, s_2)) = 0$ returns 0:

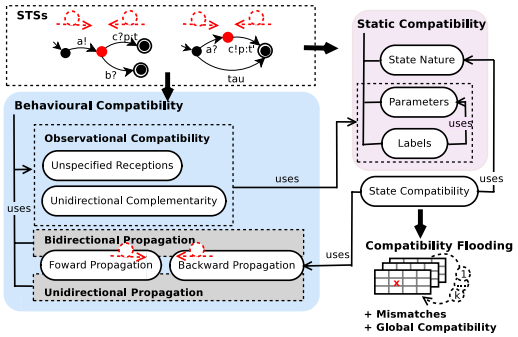


Figure 2: Compatibility Measuring Process.

Parameters. The compatibility degree of two lists of parameters exchanged with messages pl_1 and pl_2 depends on three auxiliary measures, namely: (i) the compatibility of parameter *number*, (ii) the compatibility of parameter *order*, and (iii) the compatibility of parameter *type*. These measures must be set to 1 if $pl_1 \cup pl_2 = \varnothing$ (both lists are empty) and 0 if $pl_1 \cap pl_2 = \varnothing$ (*i.e.*, both parameter lists do not share any type), respectively. Otherwise, they are computed as follows:

$$number(pl_1, pl_2) = 1 - \frac{abs(|pl_1| - |pl_2|)}{max(|pl_1|, |pl_2|)}$$

$$order(pl_1, pl_2) = 1 - \frac{|unorderedTypes(pl_1, pl_2)|}{|sharedTypes(pl_1, pl_2)|}$$

$$type(pl_1, pl_2) = 1 - \frac{|unsharedTypes(pl_1, pl_2)|}{|pl_1| + |pl_2|}$$

Here, the function $unorderedTypes$ returns the set of parameter types existing in pl_1 and pl_2—*i.e.*, shared types—but which are not in the same order in both lists. The function $unsharedTypes$ returns the set of parameter types existing in only one list.

The function *par-comp* then computes the parameter compatibility as the average of the measures returned by the three previous functions:

$$par\text{-}comp(pl_1, pl_2) = \frac{number(pl_1, pl_2) + order(pl_1, pl_2) + type(pl_1, pl_2)}{3}$$

Labels. We measure label compatibility as follows. Given a pair of labels $(l_1, l_2) \in A_1 \times A_2$ with $l_i = (m_i, d_i, pl_i)$, the function *lab-comp*(l_1, l_2) returns 0 if l_1 and l_2 have the same direction, and otherwise computes the average of the semantic compatibility of message names[4] and *par-comp*(pl_1, pl_2):

$$lab\text{-}comp(l_1, l_2) = \begin{cases} 0 & \text{if } d_1 = d_2 \\ \dfrac{sem\text{-}comp(m_1, m_2) + par\text{-}comp(pl_1, pl_2)}{2} & \text{otherwise} \end{cases}$$

2.2 Behavioural Compatibility

We now present our metrics to compute the behavioural compatibility for two service protocols, $STS_i = (A_i, S_i, I_i, F_i, T_i)$, using

[4]We assume that message names match if they are synonyms according to the Wordnet similarity package [20].

the static measures previously introduced in Section 2.1. The intuition underlying these metrics relies on the compatibility definitions given in Section 1.

We describe 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. Considering forward processing we propagate measures and mismatches effects from initial to final states and the backtracking does the inverse from the final to initial states. Hence, we can deduce the perfect compatibility or not from the initial global state. Such a propagation relies on the intuition that two states are compatible if their backward and forward neighbouring states are compatible.

The flooding algorithm returns a matrix $CM^k_{CN,D}$.[5] Each of its entries $CM^k_{CN,D}[s_1, s_2]$ stands for the compatibility measure of global state (s_1, s_2) at the k^{th} iteration. The parameter *CN* refers to the considered compatibility notion, which is checked using either an unidirectional ($D = \rightarrow$), *i.e.*, there is one service requirement which must be fulfilled by its partner, or a bidirectional ($D = \leftrightarrow$), *i.e.*, both services requirements must be fulfilled by each other, protocol traversal.

We start from an initial compatibility matrix $CM^0_{CN,D}$ where all states are supposed to be perfectly compatible, *i.e.*, $\forall (s_1, s_2) \in S_1 \times S_2,\ CM^0_{CN,D}[s_1, s_2] = 1$. In order to compute $CM^k_{CN,D}[s_1, s_2]$, we define two functions, $obs\text{-}comp^k_{CN,D}$ and $state\text{-}comp^k_{CN,D}$ detailed as follows. The first function, *observational compatibility*, computes the compatibility of outgoing and incoming observable transitions. The second function, *state compatibility*, propagates the compatibility from the forward and backward neighbouring states to (s_1, s_2), taking into account τ transitions and observational compatibility. The compatibility propagation is also parametrised according to D. In this article, we only present the forward compatibility, as the backward compatibility is handled in a similar way.

Before defining $obs\text{-}comp^k_{CN,D}$, we present a few functions necessary to its computation. Given a state $s \in S$ and a transition relation T, we define the set of emissions, receptions, and forward transitions from s as follows:

$$E(s,T) = \{t \in T \mid t = (s, (m, !, pl), s')\}$$
$$R(s,T) = \{t \in T \mid t = (s, (m, ?, pl), s')\}$$
$$Fw(s,T) = E(s,T) \cup R(s,T)$$

We let $tau(s,T) = \{t \in T \mid t = (s, \tau, s')\}$ denote the set of τ-transitions emanating from a state s. We define the function $sum^k_{CN,D}((s_1, s_2), T_1, T_2)$ as the sum of the best compatibility degree of forward neighbours of state s_1 and those of state s_2:

$$sum^k_{CN,D}((s_1, s_2), T_1, T_2) =$$
$$\begin{cases} \sum_{(s_1, l_1, s_1') \in T_1} \max_{(s_2, l_2, s_2') \in T_2} (lab\text{-}comp(l_1, l_2) \cdot CM^{k-1}_{CN,D}[s_1', s_2']) \\ \qquad \text{if } |Fw(s_1, T_1)| \neq 0 \text{ and } |(Fw(s_2, T_2)| \neq 0 \\ 0 \qquad \text{otherwise} \end{cases}$$

2.2.1 Observational Compatibility

We are now able to define the function $obs\text{-}comp^k_{CN,D}$ according to the *UR* and *UC* notions presented in Section 1.

[5] We recall that there can exist several compatibility notions *CN* in the same class D.

Unspecified Receptions. For a global state (s_1, s_2), $obs\text{-}comp^k_{UR,\leftrightarrow}$ returns 1 if there are no emissions from the states and they are deadlock free, and otherwise recursively measures the best compatibility of emissions with receptions, taking the compatibility of the states reached into account:

DEFINITION 2. *Given a global state (s_1, s_2), the observational compatibility is computed w.r.t. the UR compatibility notion as follows:*

$$obs\text{-}comp^k_{UR,\leftrightarrow}((s_1, s_2)) =$$
$$\begin{cases} 1 & \text{if } E(s_1, T_1) \cup E(s_2, T_2) = \varnothing \text{ and } (s_1, s_2) \in DF \\ 0 & \text{if } (s_1, s_2) \notin DF \\ \frac{1}{|E(s_1,T_1)|+|E(s_2,T_2)|} \cdot \Big(sum^k_{UR,\leftrightarrow}((s_1, s_2), E(s_1, T_1), R(s_2, T_2)) \\ \qquad + sum^k_{UR,\leftrightarrow}((s_2, s_1), E(s_2, T_2), R(s_1, T_1)) \Big) & otherwise \end{cases}$$

Unidirectional Complementarity. We compute how well one state s_{er} (in the *complementer* protocol) complements the state s_{ed} (in the *complemented* protocol). The comparison returns 1 if there is a subset of outgoing observable transitions in $Fw(s_{er}, T_{er})$ such that their respective labels are perfectly compatible with those of transitions in $Fw(s_{ed}, T_{ed})$. Additionally, these transitions must lead into compatible states. If there is a deadlock, then this function returns 0. Otherwise, $obs\text{-}comp^k_{UC,\rightarrow}((s_{er}, s_{ed}))$ measures the best compatibility of every transition label in $Fw(s_{er}, T_{er})$ with those in $Fw(s_{ed}, T_{ed})$, leading to the neighbouring states which have the highest compatibility degree:

DEFINITION 3. *For a global state (s_{er}, s_{ed}), the observational compatibility is computed w.r.t. the UC compatibility notion as follows, for $T'_{ed} = Fw(s_{ed}, T_{ed})$ and $T'_{er} = Fw(s_{er}, T_{er})$:*

$$obs\text{-}comp^k_{UC,\rightarrow}((s_{er}, s_{ed})) =$$
$$\begin{cases} 1 & \text{if } sum^k_{UC,\rightarrow}((s_{ed}, s_{er}), T'_{ed}, T'_{er}) = |Fw(s_{ed}, T_{ed})| \\ & \qquad \text{and } (s_{ed}, s_{er}) \in DF \\ 0 & \text{if } (s_{ed}, s_{er}) \notin DF \\ \frac{sum^k_{UC,\rightarrow}((s_{ed},s_{er}),T'_{ed},T'_{er})}{max(|T'_{ed}|,|T'_{er}|)} & otherwise \end{cases}$$

2.2.2 Directed Propagation

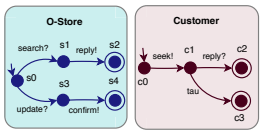
We introduce the function $fw\text{-}propag^k_{CN,D}$ which propagates the compatibility depending on the class to which *CN* belongs, *i.e.* $D \in \{\leftrightarrow, \rightarrow\}$. As far as τ transitions are concerned, this function handles also internal behaviours based upon either a $\leftrightarrow$ or $\rightarrow$ propagation.

Bidirectional Compatibility. Here, compatibility is computed from both services' point of view. That is, for a given global state (s_1, s_2), we compute the compatibility of the forward neighbours of s_1 with those of s_2 and vice-versa. For each τ transition, $fw\text{-}propag^k_{CN,\leftrightarrow}$ must be checked on the target state, and observable transitions going out of (s_1, s_2) are compared using $obs\text{-}comp^k_{CN,\leftrightarrow}$:

DEFINITION 4. *Given a global state (s_1, s_1):*

$$fw\text{-}propag^k_{CN,\leftrightarrow}(s_1, s_2) = \frac{d\text{-}fw\text{-}propag^k_{CN,\leftrightarrow}(s_1,s_2) + d\text{-}fw\text{-}propag^k_{CN,\leftrightarrow}(s_2,s_1)}{2}$$

$$d\text{-}fw\text{-}propag^k_{CN,\leftrightarrow}(s_1, s_2) =$$
$$\begin{cases} \frac{\sum_{(s_1,\tau,s_1') \in T_1} fw\text{-}propag^k_{CN,\leftrightarrow}(s_1', s_2)}{|tau(s_1,T_1)|} \\ \qquad \text{if } tau(s_1, T_1) \neq 0 \text{ and } |Fw(s_1, T_1)| = 0 \\ \frac{\sum_{(s_1,\tau,s_1') \in T_1} fw\text{-}propag^k_{CN,\leftrightarrow}(s_1', s_2) + obs\text{-}comp^k_{CN,\leftrightarrow}(s_1,s_2)}{|tau(s_1,T_1)|+1} \quad otherwise \end{cases}$$



	s0	s1	s2	s3	s4
c0	**0.78**	0.01	0.01	0.01	0.01
c1	0.01	0.68	0.01	0.35	0.01
c2	0.01	0.01	0.90	0.01	0.67
c3	0.01	0.45	0.76	0.35	0.76

Figure 3: STSs of online store system (left) and associated compatibility matrix.

Unidirectional Compatibility. To compute $\mathit{fw\text{-}propag}^k_{CN,\rightarrow}((s_1,s_2))$, from the point of view of s_2 as the complemented state, we first follow any τ-transitions from s_1, and if there are no such transitions,[6] then we follow any τ-transitions from s_2. Measuring the compatibility after every τ-transition enables us to check whether this protocol is able to fulfil its partner requirements at the target state:

DEFINITION 5. *Given a global state (s_1,s_2):*

$$\mathit{fw\text{-}propag}^k_{CN,\rightarrow}((s_1,s_2)) = \begin{cases} \frac{(\sum_{(s_1,\tau,s_1')\in T_1} \mathit{fw\text{-}propag}^k_{CN,\rightarrow}((s_1',s_2)))+\mathit{obs\text{-}comp}^k_{CN,\rightarrow}((s_1,s_2))}{|\mathit{tau}(s_1,T_1)|+1} & \text{if } \mathit{tau}(s_1,T_1)\neq\varnothing \\ \frac{(\sum_{(s_2,\tau,s_2')\in T_2} \mathit{fw\text{-}propag}^k_{CN,\rightarrow}((s_1,s_2')))+\mathit{obs\text{-}comp}^k_{CN,\rightarrow}((s_1,s_2))}{|\mathit{tau}(s_2,T_2)|+1} & \text{otherwise} \end{cases}$$

2.2.3 State Compatibility

We measure the compatibility of a global state as the weighted average of three measures, namely, forward and backward compatibility and state nature:

$$\mathit{state\text{-}comp}^k_{CN,D}(s_1,s_2) = \frac{w_1\cdot \mathit{fw\text{-}propag}^k_{CN,D}(s_1,s_2)+w_2\cdot \mathit{bw\text{-}propag}^k_{CN,D}(s_1,s_2)+\mathit{nat}(s_1,s_2)}{w_1+w_2+1}$$

where the weights w_1 and w_2 denote the number of best matches found among the outgoing and incoming, respectively, transition labels in states s_1 and s_2.

2.2.4 Compatibility Flooding

Finally, the compatibility degree of (s_1,s_2) at the k^{th} iteration is computed as the average of its previous compatibility at the $(k-1)^{th}$ iteration and the current state compatibility:

$$\mathit{CM}^k_{CN,D}[s_1,s_2] = \frac{\mathit{CM}^{k-1}_{CN,D}[s_1,s_2] + \mathit{state\text{-}comp}^k_{CN,D}((s_1,s_2))}{2}$$

EXAMPLE 1. *The table in Fig. 3 (right) shows the matrix obtained, after 7 iterations, for the example depicted according to the UC notion. Let us comment on the compatibility of states* c0 *and* s0*. The measure is quite high because both states are initial and the emission* seek! *at* c0 *perfectly matches the reception* search? *at* s0 *(they are WordNet synonyms). However, the compatibility degree is less than 1 due to the backward propagation of the deadlock from the global state* (s1, c3) *to* (s1, c1)*, and then from* (s1, c1) *to* (s0, c0)*.*

3. COMPATIBILITY FLOODING CHARACTERISTICS

[6]Given that we have excluded τ-loops, this will eventually be the case.

3.1 Convergency

In this section, we give a detailed formal proof that the computation described previously always converges to a unique compatibility matrix $\mathit{CM}_{CN,D}$. The argument is based on the fact, proven below, that the function on matrices defined in this section is a *contraction*, hence by Banach's fixed point theorem, it converges to a unique fixed point. For practical purposes, our iterative process is terminated when the distance $\varepsilon_k = |\mathit{CM}^k_{CN,D} - \mathit{CM}^{k-1}_{CN,D}|$ goes below a fixed threshold.

We show now that our algorithm for computing behavioural compatibility between services converges in a finite number of steps, regardless of its input. We show the proof for unidirectional complementarity only; it is similar for the other notions.

Let $F : [0,1]^{n\times n} \rightarrow [0,1]^{n\times n}$ be the function defined by

$$F(M)(s_1,s_2) = \frac{M(s_1,s_2) + \mathit{state\text{-}comp}_{UC,\rightarrow}(M)(s_1,s_2)}{2} \quad (1)$$

$$\mathit{state\text{-}comp}_{UC,\rightarrow}(M)(s_1,s_2) = \frac{w_1\,\mathit{fw\text{-}propag}_{UC,\rightarrow}(M)(s_1,s_2)+w_2\,\mathit{bw\text{-}propag}_{UC,\rightarrow}(M)(s_1,s_2)+\mathit{nat}(s_1,s_2)}{w_1+w_2+1} \quad (2)$$

$$\mathit{fw\text{-}propag}_{UC,\rightarrow}(M)(s_1,s_2) = \begin{cases} \frac{\sum_{(s_1,\tau,s_1')} \mathit{fw\text{-}propag}_{UC,\rightarrow}(M)(s_1',s_2) \;+\; \mathit{obs\text{-}comp}_{UC,\rightarrow}(M)(s_1,s_2)}{|tau(s_1)|+1} \\ \quad \text{if } \mathit{tau}(s_1)\neq\varnothing \\ \frac{\sum_{(s_2,\tau,s_2')} \mathit{fw\text{-}propag}_{UC,\rightarrow}(M)(s_1,s_2')+\mathit{obs\text{-}comp}_{UC,\rightarrow}(M)(s_1,s_2)}{|tau(s_2)|+1} \\ \quad \text{otherwise} \end{cases} \quad (3)$$

$$\mathit{obs\text{-}comp}_{UC,\rightarrow}(M)(s_1,s_2) = \begin{cases} 1 & \text{if } \mathit{sum}_{UC,\rightarrow}(M)(s_2,s_1,Fw(s_2),Fw(s_1)) = |Fw(s_2)| \\ & \text{and } (s_2,s_1)\in DF \\ 0 & \text{if } (s_2,s_1)\notin DF \\ \frac{\mathit{sum}_{UC,\rightarrow}(M)(s_2,s_1,Fw(s_2),Fw(s_1))}{\max(|Fw(s_2)|,|Fw(s_1)|)} & \text{otherwise} \end{cases} \quad (4)$$

$$\mathit{sum}_{UC,\rightarrow}(M)(s_1,s_2,T_1,T_2) = \begin{cases} \sum_{(s_1,l_1,s_1')\in T_1} \max_{(s_2,l_2,s_2')\in T_2} (\mathit{lab\text{-}comp}(l_1,l_2)\, M(s_1',s_2')) \\ \quad \text{if } |Fw(s_1,T_1)|\neq 0 \text{ and } |(Fw(s_2,T_2)|\neq 0 \\ 0 \quad \text{otherwise} \end{cases} \quad (5)$$

Let $\lambda = \frac{1}{2}(1+\frac{w_1+w_2}{w_1+w_2+1})$, then $\lambda < 1$. We show that F is λ-Lipschitz continuous, hence a contraction. Banach's fixed-point theorem then ensures that F has a unique fixed point and that the iteration given by equations (1) to (5) in finitely many steps reaches the fixed point with arbitrary precision. By definition of F, this fixed point is $\mathit{CM}_{UC,\rightarrow}$.

We use the max-metric for matrices, i.e. $\|M\| = \max_{i,j}|M^{ij}|$. Let $M_1, M_2 \in [0,1]^{n\times n}$, then

$$\|F(M_1)-F(M_2)\| \leq \frac{1}{2}\max_{i,j}|M_1^{ij}-M_2^{ij}| + \frac{1}{2}\|\mathit{state\text{-}comp}_{UC,\rightarrow}(M_1)-\mathit{state\text{-}comp}_{UC,\rightarrow}(M_2)\|$$

Now

$$(\textit{state-comp}_{UC,\to}(M_1) - \textit{state-comp}_{UC,\to}(M_2))^{ij} =$$
$$\tfrac{1}{w_1+w_2+1}\Big(w_1(\textit{fw-propag}_{UC,\to}(M_1)^{ij} - \textit{fw-propag}_{UC,\to}(M_2)^{ij}) + w_2(\textit{bw-propag}_{UC,\to}(M_1)^{ij} - \textit{bw-propag}_{UC,\to}(M_2)^{ij})\Big),$$

and as the formulas for $\textit{fw-propag}_{UC,\to}$ and $\textit{bw-propag}_{UC,\to}$ are entirely analogous, we can assume that

$$|(\textit{state-comp}_{UC,\to}(M_1) - \textit{state-comp}_{UC,\to}(M_2))^{ij}| \leq \tfrac{w_1+w_2}{w_1+w_2+1}|\textit{fw-propag}_{UC,\to}(M_1)^{ij} - \textit{fw-propag}_{UC,\to}(M_2)^{ij}|,$$

hence,

$$\|F(M_1) - F(M_2)\| \leq \frac{1}{2}\max_{i,j}|M_1^{ij} - M_2^{ij}| + \frac{1}{2}\frac{w_1+w_2}{w_1+w_2+1}\|\textit{fw-propag}_{UC,\to}(M_1) - \textit{fw-propag}_{UC,\to}(M_2)\|.$$

To obtain a bound for $\|\textit{fw-propag}_{UC,\to}(M_1) - \textit{fw-propag}_{UC,\to}(M_2)\|$, we note that $\textit{fw-propag}_{UC,\to}(M)(s_1, s_2)$ essentially computes a weighted average of $\textit{obs-comp}_{UC,\to}(M)(s_1', s_2')$ for all states s_1', s_2' reachable from s_1, resp. s_2, by sequences of τ-transitions: assuming $tau(s_2) = \emptyset$ for now, we have

$$\begin{aligned}
&\textit{fw-propag}_{UC,\to}(M)(s_1, s_2)\\
&= \frac{\textit{obs-comp}_{UC,\to}(M)(s_1, s_2)}{|tau(s_1)|+1} + \sum_{s_1 \xrightarrow{\tau} s_1'} \frac{\textit{fw-propag}_{UC,\to}(M)(s_1', s_2)}{|tau(s_1)|+1}\\
&= \frac{\textit{obs-comp}_{UC,\to}(M)(s_1, s_2)}{|tau(s_1)|+1} + \sum_{s_1 \xrightarrow{\tau} s_1'} \frac{\textit{obs-comp}_{UC,\to}(M)(s_1', s_2)}{(|tau(s_1)|+1)(|tau(s_1')|+1)}\\
&\quad + \sum_{s_1 \xrightarrow{\tau} s_1' \xrightarrow{\tau} s_1''} \frac{\textit{fw-propag}_{UC,\to}(M)(s_1'', s_2)}{(|tau(s_1)|+1)(|tau(s_1')|+1)} = \cdots,
\end{aligned}$$

thus,

$$\begin{aligned}
&\textit{fw-propag}_{UC,\to}(M_1)(s_1, s_2) - \textit{fw-propag}_{UC,\to}(M_2)(s_1, s_2)\\
&= \frac{\textit{obs-comp}_{UC,\to}(M_1)(s_1, s_2) - \textit{obs-comp}_{UC,\to}(M_2)(s_1, s_2)}{|tau(s_1)|+1}\\
&+ \sum_{s_1 \xrightarrow{\tau} s_1'} \tfrac{\textit{obs-comp}_{UC,\to}(M_1)(s_1', s_2) - \textit{obs-comp}_{UC,\to}(M_2)(s_1', s_2)}{(|tau(s_1)|+1)(|tau(s_1')|+1)} + \cdots +\\
&+ \sum_{s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_1^{(n)}} \tfrac{\textit{obs-comp}_{UC,\to}(M_1)(s_1^{(n)}, s_2) - \textit{obs-comp}_{UC,\to}(M_2)(s_1^{(n)}, s_2)}{(|tau(s_1)|+1)\cdots(|tau(s_1^{(n-1)})|+1)}.
\end{aligned}$$

Hence, also lifting the assumption that $tau(s_2) = \emptyset$, we see that

$$|\textit{fw-propag}_{UC,\to}(M_1)(s_1, s_2) - \textit{fw-propag}_{UC,\to}(M_2)(s_1, s_2)| \leq \max_{s_1', s_2'} |\textit{obs-comp}_{UC,\to}(M_1)(s_1', s_2') - \textit{obs-comp}_{UC,\to}(M_2)(s_1', s_2')|.$$

We have shown that

$$\|F(M_1) - F(M_2)\| \leq \frac{1}{2}\max_{i,j}|M_1^{ij} - M_2^{ij}| + \tfrac{1}{2}\tfrac{w_1+w_2}{w_1+w_2+1}\max_{i,j}|\textit{obs-comp}_{UC,\to}(M_1)^{ij} - \textit{obs-comp}_{UC,\to}(M_2)^{ij}|.$$

Now to compute an upper bound of $|\textit{obs-comp}_{UC,\to}(M_1)(s_1, s_2) - \textit{obs- comp}_{UC,\to}(M_2)(s_1, s_2)|$, we see that its maximum is attained when both values fall in the last case of (4), so

$$\begin{aligned}
&|\textit{obs-comp}_{UC,\to}(M_1)(s_1, s_2) - \textit{obs-comp}_{UC,\to}(M_2)(s_1, s_2)|\\
&= \tfrac{|\textit{sum}_{UC,\to}(M)(s_2, s_1, \textit{Fw}(s_2), \textit{Fw}(s_1)) - \textit{sum}_{UC,\to}(M)(s_2, s_1, \textit{Fw}(s_2), \textit{Fw}(s_1))|}{\max(|\textit{Fw}(s_2)|, |\textit{Fw}(s_1)|)}\\
&= \tfrac{1}{\max(|\textit{Fw}(s_2)|, |\textit{Fw}(s_1)|)} \sum_{s_2 \xrightarrow{l_2} s_2'} \Big| \max_{s_1 \xrightarrow{l_1} s_1'} \textit{lab-comp}(l_1, l_2)\, M_1(s_2', s_1) - \max_{s_1 \xrightarrow{l_1} s_1'} \textit{lab-comp}(l_1, l_2)\, M_2(s_2', s_1) \Big|\\
&\leq \tfrac{1}{\max(|\textit{Fw}(s_2)|, |\textit{Fw}(s_1)|)} \sum_{s_2 \xrightarrow{l_2} s_2'} \max_{s_1 \xrightarrow{l_1} s_1'} \textit{lab-comp}(l_1, l_2) \quad \big|M_1(s_2', s_1) - M_2(s_2', s_1)\big|\\
&\leq \frac{1}{n}\sum_j \max_i |M_1^{ij} - M_2^{ij}| \leq \max_{i,j} |M_1^{ij} - M_2^{ij}|.
\end{aligned}$$

This now entails that

$$\begin{aligned}
&\|F(M_1) - F(M_2)\|\\
&\quad \leq \frac{1}{2}\max_{i,j}|M_1^{ij} - M_2^{ij}| + \frac{1}{2}\frac{w_1+w_2}{w_1+w_2+1}\max_{i,j}|M_1^{ij} - M_2^{ij}|\\
&\quad = \lambda\|M_1 - M_2\|.
\end{aligned}$$

3.2 Mismatch Detection and Extensibility

Our compatibility measure comes with a mismatches list which identifies the incompatibility sources, *e.g.*, unmatched message names, same direction, different state natures, unordered or unshared parameter types. Regarding the compatibility matrix, for each global state we generate a list indicating the mismatch kind meaning that whether the issue is coming from state nature or deadlock or otherwise from incoming or outgoing labels (messages, directions, and parameter types). The generation of this mismatch list is done in parallel with the compatibility measuring process presented in Section 2. For instance, the states `s0` and `c1` in Fig. 3 present several mismatches, *e.g.*, `s0` is initial while `c1` is not, and their outgoing transition labels have the same directions.

Our approach is generic and can be easily extended to integrate other compatibility notions. Adding a compatibility notion *CN* only requires to define a new function $\textit{obs-comp}^k_{CN,D}$, where $D \in \{\to, \leftrightarrow\}$.

3.3 Analysis of Compatibility Measures

In this section, we first present how total protocol 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 implies that protocols are compatible if and only if their initial states are also compatible, *i.e.*, $CM_{CN,D}[I_1, I_2] = 1$. Such information is useful, *e.g.*, for automatically selecting available services in order to compose them.

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 selection step in order to find some candidates among a large number of services. Seeking for services with

Algorithm 1 *global-comp*$(S_1, S_2, CM_{CN,D}, t)$

1: *global-res* $:= 0$, *count* $:= 0$, *matched-states* $:= 0$
2: **for all** $s_1 \in S_1$ **do**
3: *match* $:=$ *False*
4: **for all** $s_2 \in S_2$ **do**
5: **if** $CM_{CN,D}[s_1, s_2] \geq t$ **then**
6: *global-res* $:=$ *global-res* $+ CM_{CN,D}[s_1, s_2]$
7: *match* $:=$ *True*; *count* $:=$ *count* $+ 1$
8: **if** *match* $=$ *True* **then**
9: *matched-states* $:=$ *matched-states* $+ 1$
10: **if** *count* $\neq 0$ **then**
11: *global-res* $:= \frac{\textit{global-res}}{\textit{count}} \cdot \frac{\textit{matched-states}}{|S_1|}$
12: **return** *global-res*

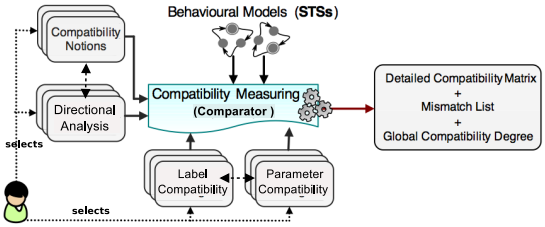


Figure 4: Comparator architecture.

high global compatibility degree enables to simplify further processing to resolve their interface incompatibility, *e.g.*, using service adaptation [12].

The global compatibility can be computed differently depending on the user preferences. A first solution consists in computing the average of the maximal compatibility degrees computed for all states. An alternative, shown in Algorithm 1, is to compute the global compatibility degree as the average of all compatibility degrees that are higher than or equal to a threshold t. To account for unmatched states, we multiply this average by the rate of states which have at least one possible matching with compatibility degree higher than t.

Algorithm 1 computes the global compatibility measure from one STS's point of view, and this works for the unidirectional compatibility notions. For the bidirectional compatibility notions, the global compatibility is computed as the average of the values returned by both functions *global-comp*$(S_1, S_2, CM_{CN,D}, t)$ and *global-comp*$(S_2, S_1, CM_{CN,D}, t)$.

EXAMPLE 2. *This example illustrates the computation of the global compatibility degree for the online store system of Fig. 3. Given a threshold* $t = 0.7$ *and the matrix of Table 3, the application of Algorithm 1 returns a global compatibility degree of* 0.6. *This rather low measure is justified by the state mismatch at* c1 *of the* Customer *protocol, which does not match with any state of the* O-Store *protocol (all compatibility values are below the threshold).*

We can now also finish our example from the introduction, *cf.* Fig. 1. As this is a client-server setting, we compute compatibility using the *UR* notion. Using our tool Comparator and a threshold of 0.7 as above, we can compute the three MedServers' compatibility degrees with the Client service to 0.55, 0.85, and 0.76, respectively, hence preferring MedServer1 over the other two.

4. TOOL SUPPORT

4.1 Implementation

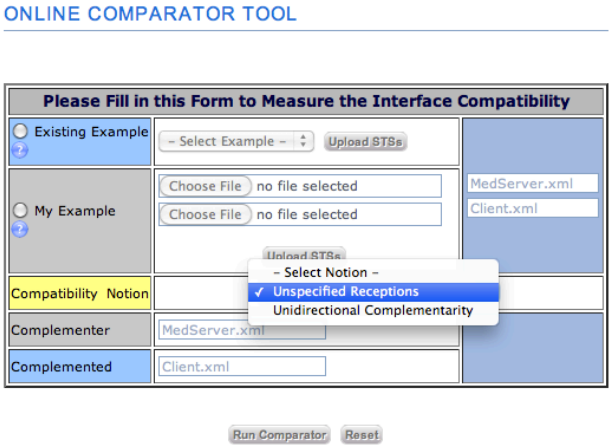


Figure 5: Online Comparator: Online User Interface.

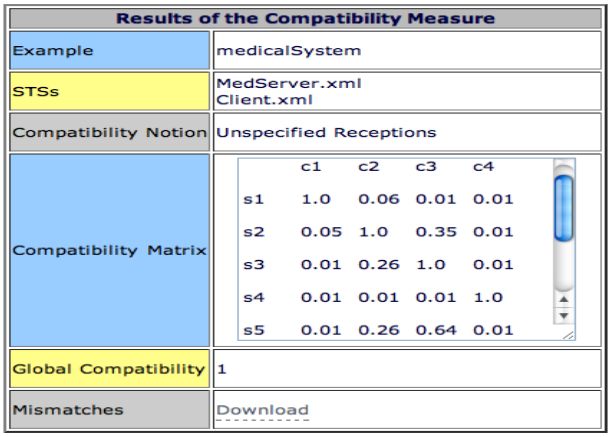


Figure 6: Online Comparator: Result.

Our approach for measuring the compatibility degree of service protocols has been fully implemented in a prototype tool called Comparator [5]. The framework architecture is given in Fig. 4. The tool, implemented in Python, 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. The tool returns the compatibility matrix, the mismatch list, and the global compatibility degree which indicates how compatible both services are. The implementation is highly modular, which makes easy its extension with new compatibility notions or other strategies for comparing message names and parameters.

Our Comparator tool can be used through a Web application [5] which enables the users to check either some examples from our database or their own examples that they can upload to our application. Figures 5 and 6 illustrate with a couple of screenshots the use of our application. The first one represents the user interface to set the measuring parameters—example, compatibility notion, and roles in the case of *UC* notion. The second Figure shows an example of compatibility measure computed using Comparator.

4.2 Experimental Results

We have validated our prototype tool on more than 110 examples,

ranging from small ones, to experiment boundary cases, to real-world examples, *e.g.*, a car rental [6], a travel booking system [12], a video-on-demand application [21], music player system [21], a medical management system [4], and a multi-function device service [22]. For illustration purposes, three case studies are available online at [5] and present the results of our approach for quantifying the compatibility.

Table 1 summarises the results of some of the examples of our database. Experiments have been carried out on a Mac OS machine running on a 2.53 GHz Intel dual core processor with 4 GB of RAM. The computation time differs with respect to the input interface size (states and transitions). Experiments show that small examples with few states and transitions (*e.g.*, Ex9, Ex44, Ex71) require a negligible time for measuring their compatibility, whereas bigger examples (*e.g.*, Ex90, Ex101) need more time (see Table 1). The computation time increases with respect to the number of τ branchings and loops. For instance, the size of Ex85 is quite big but this example consists of protocols with sequential structure and including very few loops, therefore the computation time does not exceed two minutes. On the other hand, the conducted experiments showed that protocols involving many loops (*e.g.*, Ex90) require more time than those having only few loops (*e.g.*, Ex85). To sum up, experiments have shown that Comparator computes the compatibility degree of quite large systems (*e.g.*, services with hundreds of states and transitions) in a reasonable time (many iterations are performed in a few minutes).

It is worth noticing that efficiency was not our main concern in the current implementation of Comparator. Our experiments (as sketched in Table 1) aim at illustrating our approach on concrete examples and at showing that it computes the compatibility degree of service protocols in a reasonable time.

4.3 Evaluation

Human Expert Evaluation. The first author has been considered an expert to compare the measure automatically computed with the manual analysis of service compatibility and mismatches. This study has shown that each time a couple of states in two protocols presents several mismatches according to the manual evaluation, this corresponds to a low value in the matrix and vice-versa.

We also realised that the manual analysis of interaction protocols is time-consuming and confusing, specially in case of large and complex systems. However, the comparator tool performs many iterations to compute the compatibility of large systems—considering many loops and internal behaviours—in a systematic and automating way, saving time and human effort. Our approach is valuable, it enables a quick and reasonable comparison compared with the manual processing.

Although it can be relatively easy to compare some small protocols by hand, protocol verification is hard and tedious issue, specially for users who are not familiar with the verification issue. In such a case, manual processing is time-consuming, error-prone and always likely to be laborious. The conducted experiments proved that our automatic analysis covers the aforementioned issues. The resulting matrix and detected mismatch list have always justified the interface incompatibility.

Accuracy. To evaluate the preciseness of our compatibility measure, we reuse the well-known precision and recall metrics [23] to estimate how much the measure automatically computed meets the expected result (see [5] for more details). 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_1, s_2) is a correct match if the state $s_1 \in S_1$ has the highest compatibility degree with $s_2 \in S_2$ among those in S_2. Our measuring process yields a precision and recall of 100% for compatible protocols. Our empirical analysis has shown the good quality of our approach for comparing incompatible protocols. For instance, the study of the car rental system [6] 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 [5] which helps travellers 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 both metrics.

Application. Some researchers from GISUM laboratory[7] have applied the result computed by Comparator for service adaptation purpuses [3]. Our tool enables a quick compatibility measuring and gives to the user a detailed interface comparison. This was a valuable help to resolve interface mismatches and suggest adaptation solutions.

5. RELATED WORK

We present several related approaches to measuring similarity or compatibility of interfaces. These are applied for service substitution and composition, respectively.

Protocol Traversal. The work in [24] measures the similarity of two computer viruses described using labelled transition systems (LTSs). It uses quantitative functions which are computed by a simple (not iterative) forward traversal of two LTSs. This work does not return the differences which distinguish one service from another, and there is no computation of a global similarity degree. In [26], the authors check the compatibility of two services described using the π-calculus. Here, two services are compatible if there is always at least one transition sequence between them, until reaching final states. This notion is too weak since it does not guarantee deadlock-freeness for service composition. The authors compute the compatibility degree of two services as the average of the number of successful transition sequences. Neither detailed compatibility of different protocol states nor the mismatch list is returned.

Edit Distance. In [11, 1], the authors calculate the edit distance between a given *defective* service and *synthesised correct* services. They also detect the differences between two versions of one service interface described using finite state machines. The quantitative simulation measures the state similarity based on the analysis of outgoing transition labels without any semantic comparison of these label names, and there is no propagation of compatibility between neighbouring states.

Similarity Flooding. In [13, 14], the similarity flooding technique was applied to the problem of model matching. This algorithm returns a matrix for the similarity propagation which is updated iteratively. The authors propose a set of metrics to measure correspondences between elements of data structures such as data schemas or data instances, described with LTSs. The work in [13] aims at assisting developers in matching elements of a schema by suggesting candidates. However, their tool does not enable fully automated matching. In [14], 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, [15] proposes a semi-automated approach for checking the matching of messages in two business

[7] http://www.gisum.uma.es

Table 1: Some Experimental Results ($t = 0.7$; last column shows threshold).

Example	States	Transitions	Compatibility Notion	*global*	Time (mn)	Iterations (k)	ε
Ex9	8/5	8/5	*UR*	0.29	0m0.415s	8	0.01
			UC	0.18	0m10.581s	8	0.01
Ex44	20/22	19/21	*UR*	1	0m4.440s	8	0.02
			UC	0.81	0m13.860s	3	0.19
Ex71	20/4	19/3	*UR*	1	0m4.112s	8	0.02
			UC	1	0m5.848s	9	0.01
Ex85	59/59	64/75	*UR*	0.70	1m1.717s	4	0.13
			UC	0.69	0m47.513s	3	0.17
Ex90	86/86	90/90	*UR*	0.72	8m15.806s	7	0.04
			UC	0.74	2m48.400s	3	0.19
Ex101	124/86	135/90	*UR*	0.69	19m0.575s	10	0.02
			UC	0.70	8m0.460s	6	0.13

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.

Quantitative Model Checking. The quantitative approach to service compatibility which we advocate here is related to recent quantitative approaches to model checking and verification. Here, Boolean notions of verification are replaced by distances, just as we do here for service compatibility. A general framework for such distance-based quantitative verification has been developed in [9, 8, 10].

6. CONCLUSION

This paper proves the convergency of our flooding algorithme for measuring the compatibility degree between behavioural models. Our proposal is fully supported by the standalone Comparator tool which has been validated on many examples.

Our work has straightforward applications to service-related issues, *e.g.*, automatic discovery, selection, ranking, and composition. On a wider scale, our solution is useful for all interactive software systems described using STSs. In particular, Comparator has been used in a real-world case study in the context of the ITACA project [3] for service composition and semi-automated adaptation. Comparator has also been integrated into a prototype tool, called Updator [18], which we implemented to deal with service evolution issues. Our main perspective is to to apply our compatibility measuring approach for the automatic generation of adaptor protocols.

7. REFERENCES

[1] A. Aït-Bachir. Measuring similarity of service interfaces. In *ICSOC PhD Symposium*, volume 421 of *CEUR Workshop Proceedings*, 2008.

[2] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky, and M. Tivoli. Towards an engineering approach to component adaptation. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 193–215. Springer, 2006.

[3] J. Cámara, J. A. Martín, G. Salaün, J. Cubo, M. Ouederni, C. Canal, and E. Pimentel. Itaca: An integrated toolbox for the automatic composition and adaptation of web services. In *ICSE'09*, pages 627–630. IEEE, 2009.

[4] J. Cámara, G. Salaün, C. Canal, and M. Ouederni. Interactive specification and verification of behavioural adaptation contracts. In *QSIC'09*, pages 65–75. IEEE, 2009.

[5] Comparator, , http://ouederni.perso.enseeiht.fr/tools.html.

[6] J. Cubo, G. Salaün, C. Canal, E. Pimentel, and P. Poizat. A model-based approach to the verification and adaptation of WF/.NET components. In *FACS*, volume 215 of *ENTCS*, pages 39–55, 2008.

[7] F. Durán, M. Ouederni, and G. Salaün. A generic framework for n-protocol compatibility checking. *Sci. Comput. Program.*, 77(7-8):870–886, 2012.

[8] U. Fahrenberg and A. Legay. Generalized quantitative analysis of metric transition systems. In *APLAS*, volume 8301 of *LNCS*, pages 192–208. Springer, 2013.

[9] U. Fahrenberg and A. Legay. The quantitative linear-time-branching-time spectrum. *Theor. Comput. Sci.*, 538:54–69, 2014.

[10] U. Fahrenberg, A. Legay, and C. Thrane. The quantitative linear-time–branching-time spectrum. In *FSTTCS*, volume 13 of *LIPIcs*, pages 103–114. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[11] N. Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *BPM'08*, volume 5240 of *LNCS*, pages 132–147. Springer, 2008.

[12] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. *IEEE Trans. Software Eng.*, 38(4):755–777, 2012.

[13] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128. IEEE, 2002.

[14] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *ICSE'07*, pages 54–64. ACM Press, 2007.

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

[16] M. Ouederni and G. Salaün. Tau be or not tau be? - a perspective on service compatibility and substitutability. In *WCSI'10*, volume 37 of *EPTCS*, pages 57–70, 2010.

[17] M. Ouederni, G. Salaün, and E. Pimentel. Quantifying service compatibility: A step beyond the boolean approaches. In *ICSOC'10*, volume 6470 of *LNCS*, pages 619–626, 2010.

[18] M. Ouederni, G. Salaün, and E. Pimentel. Client update: A solution for service evolution. In *SCC'12*, pages 394–401. IEEE, 2011.

[19] M. Ouederni, G. Salaün, and E. Pimentel. Measuring the

compatibility of service interaction protocols. In *SAC'11*, volume 2, pages 1560–1567. ACM, 2011.

[20] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *AAAI'04*, pages 1024–1025. AAAI, 2004.

[21] P. Poizat, G. Salaün, and M. Tivoli. An Adaptation-based Approach to Incrementally Build Component Systems. *ENTCS*, 182:155–170, 2007.

[22] G. Salaün. Generation of Service Wrapper Protocols from Choreography Specifications. In *SEFM'08*, pages 313–322. IEEE, 2008.

[23] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

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

[25] R. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

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

[27] D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.