HAL
open science

Comparator: A Tool for Quantifying Behavioural Compatibility

Meriem Ouederni, Gwen Salaün, Javier Cámara, Ernesto Pimentel

To cite this version:

Meriem Ouederni, Gwen Salaün, Javier Cámara, Ernesto Pimentel. Comparator: A Tool for Quantifying Behavioural Compatibility. FASE 2014 - 17th International Conference on Fundamental Approaches to Software Engineering, Apr 2014, Grenoble, France. pp.306-309, 10.1007/978-3-642-54804-8_21 . hal-00934057

HAL Id: hal-00934057

https://inria.hal.science/hal-00934057

Submitted on 21 Jan 2014

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.

Comparator: A Tool for Quantifying Behavioural Compatibility

Meriem Ouederni[1], Gwen Salaün[2], Javier Cámara[3], and Ernesto Pimentel[4]

[1] Toulouse INP, IRIT, France
[2] Grenoble INP, Inria, France
[3] Institute for Software Research, Carnegie Mellon University, USA
[4] Department of Computer Science, Universidad de Málaga, Spain

Abstract. We present Comparator, a tool that measures the compatibility between two behavioural interfaces. Comparator can be used as a stand-alone Web application, and is also integrated into a model-based adaptation toolbox.

1 Introduction

Context. Building new applications by composing existing software components or Web services is now mainstream. However, this task remains error-prone, especially when reusing stateful components accessed through their behavioural interfaces. Techniques and tools are therefore necessary to support this composition task, and to make sure that the new system will behave correctly, avoiding undesired behaviours such as deadlocks.

Model. In this work, we assume that component 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). In particular, a STS is a tuple (A, S, I, F, T) where A is an alphabet which corresponds to the set of labels, S is a set of states, $I \in S$ is the initial state, $F \subseteq S$ is a nonempty set of final states, and $T \subseteq S \times A \times S$ is the transition relation. Note that a *label* is either the (internal) τ action or a tuple (m, d, pl) where m is the message name, d indicates 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. STSs can be easily derived from higher-level description languages such as Abstract BPEL for instance where such abstractions were used for verification, composition or adaptation of Web services.

Contributions. In this tool paper, we present Comparator, a tool supporting the composition task by analysing the behavioural interfaces of the components to be composed. Comparator accepts as input two behavioural interfaces described using STSs. We assume that both STSs interact *wrt.* a synchronous communication model. Our tool indicates whether the interfaces can interoperate correctly. Otherwise, it provides three outputs: a detailed compatibility measure for all states in both STSs, a list of mismatches, and a global compatibility measure.

Comparator can be used as a stand-alone application through a Web interface. It is also integrated into ITACA [2], a toolbox for model-based adaptation.

2 Quantifying Behavioural Compatibility

Interfaces are compatible if they interact successfully with no mismatch *wrt.* a criterion set on their observable actions. This criterion is called compatibility notion, *e.g.*, *unspecified receptions* where all reachable emissions can be received in the other STS, and *unidirectional complementarity* where all actions in one STS have a matching in the other STS [3].

In this section, we overview the main ideas behind our measure computation. All the theoretical background for identifying possible mismatches and measuring the compatibility of two STSs is presented in [4]. The computation process accepts as input two protocols $STS_1 = (A_1, S_1, I_1, F_1, T_1)$ and $STS_2 = (A_2, S_2, I_2, F_2, T_2)$ and computes a compatibility degree for each global state, *i.e.,* each couple of states (s_1, s_2) with $s_1 \in S_1$ and $s_2 \in S_2$. All compatibility scores range between 0 and 1, where 1 means a perfect compatibility. Our approach is parameterised by a compatibility notion, that is, we measure how far the two interfaces are from being compatible *wrt.* this compatibility notion.

To measure the compatibility of two STSs, we compute the compatibility degree for all possible global states in two steps. We first compute a static compatibility based on the comparison of state nature (*i.e.*, initial, final, or none of them), labels, and types of exchanged parameters. These measures are then used to quantify the behavioural compatibility taking the label ordering into account and the structure of both STSs. The second step returns the compatibility measure for all global states in both STSs. State compatibility is based on the fact that two states are compatible if their preceding and succeeding neighbouring states are compatible, where the preceding and succeeding neighbours of state s' in transitions (s, l, s') and (s', l', s'') are respectively the states s and s''. Hence, in order to measure the compatibility degree of two protocols, we consider an iterative approach which propagates the compatibility degree from one state to all its neighbours. This process is called compatibility flooding and works using a double propagation (forward and backward).

3 Online Comparator Tool

Our approach for measuring the protocol compatibility degree has been fully implemented in a tool called Comparator. We encoded it in Python 2.6 using Eclipse 3.5.1 as programming IDE. The tool accepts as input two XML files corresponding to the interfaces, and a compatibility notion used as comparison criterion. Comparator returns the compatibility matrix, the mismatch list, and the global compatibility degree, which indicates how compatible the two interfaces are. The implementation of our proposal is highly modular, thus facilitating its extension with other compatibility notions. In order to make our Comparator tool widely

available to any potential users, we implemented a Web interface [1] so that anyone can use and run it online (Fig. 1).

Experimental Results. We validated our tool on about 110 real-world examples, *e.g.*, a car rental service, a travel booking system, a medical management system, or an online email service. Some of these examples are available online [1] to illustrate the results returned by our compatibility measure. Note that Comparator computes the compatibility degree of quite large systems (*e.g.*, interfaces with hundreds of states and transitions) in a reasonable time (a few minutes).

Evaluation. We evaluate our tool accuracy using precision and recall metrics [5], which estimate how much our measure meets the expected result. Precision measures the matching quality (number of false positive matches) and is defined as the ratio of the number of correct state matches found out of the total of state matches found. Recall is the coverage of the state matching results and is defined as the ratio of the number of correct state matches found out of the total of all correct state matches in the two protocols. An effective measure must produce high precision and recall values. We have computed these metrics for the examples of our database using both UC and UR notions. 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 also showed the good quality of our approach for comparing incompatible protocols. For instance, the study of the car rental service [1] produces a precision and recall equal to 85% and 95%, respectively. We applied the same evaluation to a flight advice system [1] which helps travellers to find flight information. This yields a precision and a recall equal to 91% and 100%, respectively. We measured precision and recall for the other examples of our dataset as well, and our study revealed very high values for both metrics (more than 90% in average).

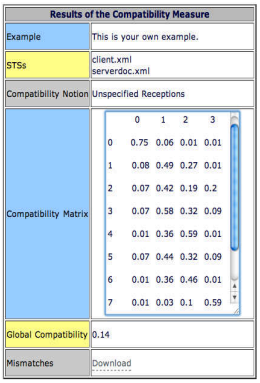
4 Application to Model-based Adaptation of Web Services

Our compatibility degree results have some straightforward applications for, *e.g.*, service selection, ranking, and adaptation. We focus in the rest of this section on software adaptation [6]. Adaptation aims at computing an intermediate component or *adaptor* to resolve mismatches existing between services interacting with each other. An adaptor is built from abstract descriptions, *a.k.a* adaptation contracts, specifying how the involved services can successfully interact together for fulfilling some specific requirements in spite of the mismatches existing in their interfaces. The Comparator tool was integrated into a graphical environment, called ACIDE, for the interactive specification of adaptation contracts. This module belongs to a complete framework, called ITACA, dedicated to the design and synthesis of adaptors for Web services [2].

ACIDE includes a graphical representation of STSs and a visualization of their *ports*. Each label on the STS corresponds to a port in the graphical description. Ports include a data port for each parameter contained in the parameter list of

the label. Correspondences between STSs are represented as port bindings and data port bindings. Starting from the graphical representation, the architect can specify these bindings by successively connecting ports and data ports. The resulting collection of bindings is the adaptation contract.

Our compatibility measure can be used in different ways to specify the adaptation contract in ACIDE. Firstly, it is possible to automatically generate port bindings for labels that perfectly match. Secondly, the designer can also select a state (label, resp.) in one protocol, and Comparator returns the best state (label, resp.) matching in the other protocol. For instance, Fig. 2 shows the state-based matching results when the designer selects state number 2 in the top left STS and compares it with all the states in the client STS on the right.



Results of the Compatibility Measure	
Example	This is your own example.
STSs	client.xml serverdoc.xml
Compatibility Notion	Unspecified Receptions
Compatibility Matrix	
Global Compatibility	0.14
Mismatches	Download

	0	1	2	3
0	0.75	0.06	0.01	0.01
1	0.08	0.49	0.27	0.01
2	0.07	0.42	0.19	0.2
3	0.07	0.58	0.32	0.09
4	0.01	0.36	0.59	0.01
5	0.07	0.44	0.32	0.09
6	0.01	0.36	0.46	0.01
7	0.01	0.03	0.1	0.59

Fig. 1. Online Comparator

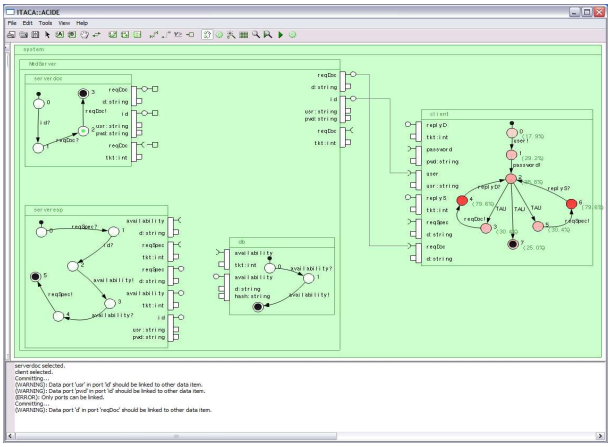


Fig. 2. State-based matching in ACIDE

References

1. Comparator Web Page. http://ouederni.perso.enseeiht.fr/10-comparator-tool.html.
2. J. Cámara, J. Antonio 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 *Proc. of ICSE'09*, pages 627–630. IEEE, 2009.
3. F. Durán, M. Ouederni, and G. Salaün. A Generic Framework for N-Protocol Compatibility Checking. *SCP*, 77(7-8):870–886, 2012.
4. M. Ouederni, G. Salaün, and E. Pimentel. Measuring the Compatibility of Service Interaction Protocols. In *Proc. of SAC'11*, pages 1560–1567. ACM, 2011.
5. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill New York, 1983.
6. D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.