ITACA: An Integrated Toolbox for the Automatic Composition and Adaptation of Web Services

Javier Cámara, José Antonio Martín, Gwen Salaün,
Javier Cubo, Meriem Ouederni, Carlos Canal, Ernesto Pimentel
Department of Computer Science, University of Málaga, Spain
{jcamara,jamartin,salaun,cubo,meriem,canal,ernesto}@lcc.uma.es

Abstract

Adaptation is of utmost importance in systems developed by assembling reusable software services accessed through their public interfaces. This process aims at solving, as automatically as possible, mismatch cases which may be given at the different interoperability levels among interfaces by synthesizing a mediating adaptor. In this paper, we present a toolbox that fully supports the adaptation process, including: (i) different methods to construct adaptation contracts involving several services; (ii) simulation and verification techniques which help to identify and correct erroneous behaviours or deadlocking executions; and (iii) techniques for the generation of centralized or distributed adaptor protocols based on the aforementioned contracts. Our toolbox relates our models with implementation platforms, starting with the automatic extraction of behavioural models from existing interface descriptions, until the final adaptor implementation is generated for the target platform.

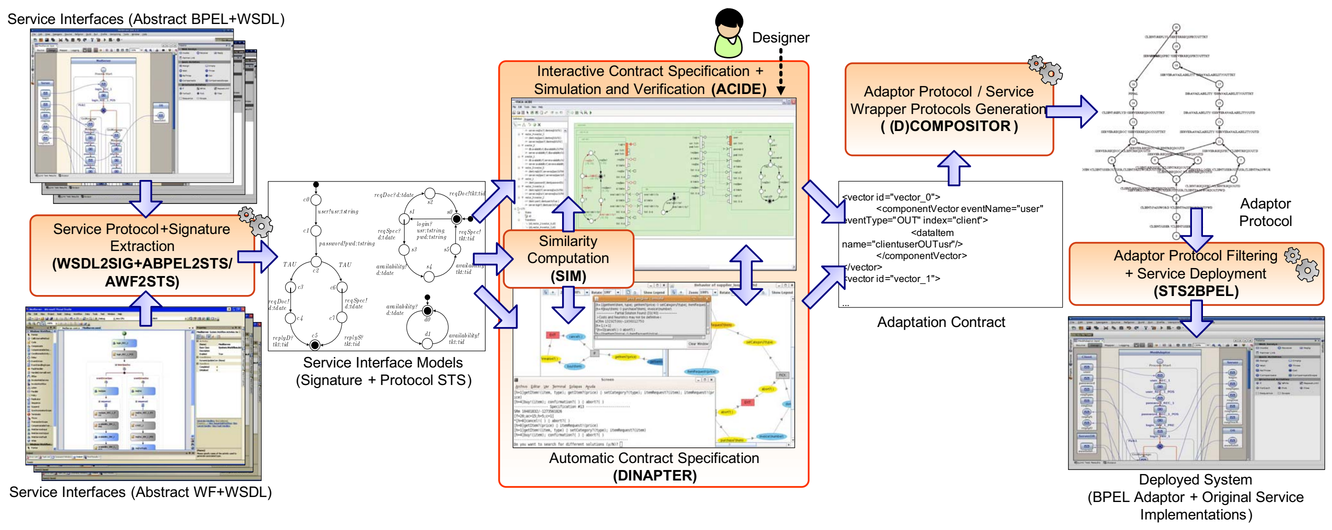
1 Introduction

In today's Web, services are everywhere. These can be just accessed and used to fulfill basic requirements, or be composed with other services to build bigger systems which aim at working out complex tasks. To ease their reuse and enable their automatic composition, services must be equipped with rich interfaces enabling external access to their functionality. Several interoperability levels can be distinguished in interface description languages (*i.e.,* signature, protocol, quality of service, and semantics). Composition of services is seldom achieved seamlessly because mismatch may occur at the different interoperability levels and must be solved. *Software adaptation* is the only way to compose non-intrusively black-box components or services with mismatching interfaces by automatically generating mediating *adaptor* services.

So far, most adaptation approaches have assumed interfaces described by signatures (operation names and types) and behaviours (interaction protocols). Describing protocol in service interfaces is essential because erroneous executions or deadlock situations may occur if the designer does not consider them while building composite services. Deriving adaptors is a complicated task since, in order to avoid undesirable behaviours, the different behavioural constraints of the composition must be respected, and the correct execution order of the messages exchanged must be preserved while mismatch is corrected.

Most existing works on model-based behavioural adaptation (see for instance [2, 4, 15]) favour the full automation of the process. They are referred to as *restrictive approaches* because they try to solve interoperability issues by pruning the behaviours that may lead to mismatch, thus restricting the functionality of the services involved. These techniques are limited since they are not able to fix subtle incompatibilities between service protocols by remembering and reordering events and data when necessary. A second class of solution is referred to as *generative approaches* (see for instance [3, 5, 7]). These avoid restricting service behaviour, and support the specification of advanced adaptation scenarios. Generative approaches build adaptors automatically from an abstract specification, namely an *adaptation contract*, of how mismatch cases can be solved.

In this paper we present ITACA [1], an integrated toolbox that fully supports generative adaptation (see Fig. 1 for an overview of the process), which starts with the automatic extraction of behavioural models from existing interface descriptions either in Abstract BPEL or Windows Workflows (WF). The toolbox enables automatic contract generation, and also interactive contract specification. The latter relies on a graphical notation and a computation of protocol similarity which assists the designer by pointing out parts of service protocols that can be matched together. Interactive specification can be used either as an alternative, or in conjunction with automatic generation. Simulation and verification of the system's execution based on a particular con-


Service Interfaces (Abstract BPEL+WSDL)
Service Protocol+Signature Extraction (WSDL2SIG+ABPEL2STS/ AWF2STS)
Service Interfaces (Abstract WF+WSDL)
Service Interface Models (Signature + Protocol STS)
Designer
Interactive Contract Specification + Simulation and Verification (ACIDE)
Similarity Computation (SIM)
Automatic Contract Specification (DINAPTER)
Adaptor Protocol / Service Wrapper Protocols Generation ((D)COMPOSITOR)
Adaptation Contract
Adaptor Protocol
Adaptor Protocol Filtering + Service Deployment (STS2BPEL)
Deployed System (BPEL Adaptor + Original Service Implementations)

```
<vector id="vector_0">
        <componentVector eventName="user"
eventType="OUT" index="client">
                <dataItem
name="clientuserOUTusr"/>
        </componentVector>
</vector>
<vector id="vector_1">

...
```

Figure 1. Adaptation process overview in ITACA

tract before adaptor generation is also possible. Finally, a monolithic adaptor protocol or a set of distributed adaptation wrapper protocols can be automatically generated and deployed.

2 Overview of the Toolbox

2.1 Behavioural Interfaces

We assume that service interfaces are specified using both a signature and a protocol. Signatures correspond to operation names associated with arguments and return types relative to the messages and data being exchanged when the operation is called. Protocols are represented by means of Symbolic Transition Systems (STSs), which are Labelled Transition Systems (LTSs) extended with value passing [17]. This formal model has been chosen because it is simple, graphical, and provides a good level of abstraction to tackle verification, composition, or adaptation issues [8, 9, 18]. At the user level, one can specify service interfaces (signatures and protocols) using respectively WSDL, and Abstract BPEL (ABPEL) or WF workflows (AWF) [6].

WSDL2SIG. This tool parses WSDL descriptions of Web services and generates the corresponding signatures.

ABPEL2STS/AWF2STS. This parser generates STSs from service interfaces specified using ABPEL or AWF.

To ease the addition of other possible notations to describe service interfaces, we use as an intermediate step in this parsing process an abstract Web services class (AWS). Thus, one can add as a front-end another description language with its parser to AWS, and take advantage of the existing parser from AWS to our model (see Fig. 2).

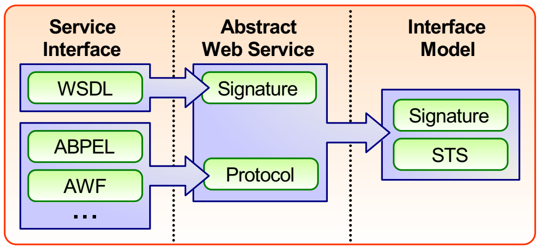


Figure 2. Behavioural interface extraction

2.2 Adaptation Contract Specification

In this section, we introduce specification techniques for adaptation contracts. A contract matches operations required by service interfaces with those offered by others in order to reconcile interface mismatch at the signature and behavioural levels.

SIM. Understanding how two protocols differ helps to build adaptation contracts by suggesting the best possible operation matches to the user. To do so, we have implemented a tool, SIM [16], which measures protocol similarity. Such measure aims at pointing out differences between two protocols, but also at detecting parts of them which turn out to be similar. Our tool relies on a *divide-and-conquer* approach to compute the similarity of service protocols (described as STSs) from a set of four detailed similarity comparisons, namely for states, labels, depths and graphs. This information can be used to guide the contract specification process.

Dinapter. In order to alleviate the cumbersome task of designing adaptation contracts and to avoid mistakes in the specification (which may lead to undesirable behaviours of the system), we implemented a tool aimed at the automatic generation of contracts called Dinapter [12]. This tool traverses the behaviour of the services and matches the operations found based on the metrics returned by SIM. Dinapter exploits these metrics to match compatible operations and to adapt the minimum set of operations required for the deadlock-free composition of services. The generated contracts successfully specify how to overcome signature mismatch (*i.e.*, different operation names and arguments) and behavioural incompatibilities (*i.e.*, message splitting/merging, missing messages and message reordering) in such a way that all services are able to interact with each other and reach a final state.

ACIDE. Automatic contract generation may produce solutions leading to deadlock-free compositions unable to fulfill their intended goals. Therefore, ITACA incorporates an Adaptation Contract Interactive Design Environment, which aims at helping the designer in specifying a contract, reducing the risk of errors introduced by manual specification. In contrast with using textual notations where the designer can write any (correct or incorrect) statement, our tool uses a graphical notation which enables interactive and incremental construction and checks on the contract. Thus, any contract produced with the tool is syntactically correct and consistent. In addition, the interactive environment is able to:

– Assist the designer by pointing out the best matches between ports graphically using protocol similarity information obtained from the SIM tool.

– Simulate the execution of the system step-by-step and determine how the different behavioural interfaces evolve as the different parts of the contract are executed, highlighting active states and fired transitions on the graphical representation of interfaces. This helps the designer to detect the behavioural issues that might be raised during execution and to understand if the behaviour of the system complies with his/her design intentions.

– Automatically identify execution traces leading to deadlock or livelock. These can be replayed step-by-step using simulation to understand the cause of the incorrect behaviour.

It is worth observing that Dinapter and ACIDE mutually improve their results when they are combined. On one hand, when Dinapter receives adaptation constraints from the interactive design environment, it is able to discard solutions leading to deadlock-free compositions that may not fulfill their intended goals (*e*.g., a client-supplier system which always aborts requests). On the other hand, the designer can select parts of a contract automatically constructed by Dinapter through the ACIDE environment.

2.3 Adaptor and Wrapper Generation

From a set of service protocols and a contract specification, one can generate either an *adaptor* protocol (centralized view), or a set of *adaptation wrapper* protocols (distributed view). In the first case, the adaptor can be deployed on a single machine. In the case of wrappers, they can be distributed and deployed using middleware technologies, preserving a full parallelism of the system's execution. Adaptor and wrapper protocols are automatically generated in two steps: (i) system's constraints are encoded into the LOTOS [11] process algebra, and (ii) adaptor and wrapper protocols are computed from this encoding using on-the-fly exploration and reduction techniques. The reader interested in more details may refer to [13, 17].

(D)Compositor. This tool generates LOTOS code for service interfaces, the adaptation contract, and some processes indicating how to obtain distributed wrappers. Beyond simulation and verification techniques integrated in ACIDE, the LOTOS encoding allows to check temporal logic properties on the adaptor under construction using the CADP model-checker, Evaluator [14].

Scrutator. This tool belongs to the CADP toolbox [10], and returns adaptor and wrapper protocols (STSs) corresponding to the LOTOS specification generated by (D)Compositor. It removes remaining erroneous paths, τ transitions and path similarities by applying state-of-the-art exploration and reduction techniques while avoiding the full state space generation corresponding to the LOTOS specification. Adaptor and wrapper protocols are platform-independent and we show in the next section how they can be refined *wrt.* a specific platform.

2.4 Implementation

Our internal model (STS) can take into account some additional behaviours (interleavings) that cannot be implemented into executable languages (*e.g.,* BPEL). To make platform-independent adaptor protocols implementable *wrt.* a specific platform, some filters are used with the purpose of pruning parts of the protocol corresponding to these interleavings and keep only executable paths.

STS2BPEL. In particular, to implement an adaptor model as a BPEL orchestrator we proceed in two steps: (i) filtering the interleaving cases that cannot be implemented (*e.g.,* several emissions and receptions outgoing from a same state), and (ii) encoding the filtered model into the corresponding implementation language. Following the guidelines presented in [13], the adaptor protocol is implemented using a state machine pattern. The main body of the BPEL process corresponds to a global *while* activity with *if* statements used inside it to encode adaptor states. Each *if* body encodes transitions outgoing from the corresponding state. Variables are used to store data passing through the adaptor and the current state of the protocol.

3 Conclusions

Building systems by adapting a set of reusable software services is an error-prone task which can be supported by assisting developers with automatic procedures and tools. In this work, we have presented what is, to the best of our knowledge, the first toolbox that fully supports a generative adaptation approach from beginning to end. ITACA supports the specification and verification of adaptation contracts, automates the generation of adaptor protocols, and relates our abstract models with implementation languages.

ITACA has been implemented in Python and Java, and consists of about 51,000 lines of code. We have intensively applied and validated our toolbox on many examples, which range from synthetic ones used to experiment boundary cases, to real-world case studies such as a travel agency, rate finder services, on-line computer material store, library management systems, SQL servers, and many other systems. Although our toolbox automates all the steps of the adaptation process, contract specification may require human intervention to ensure that the goal of the composition is fulfilled. However, experiments we have carried out show that, even in this case, the techniques proposed in ITACA to support the adaptation contract construction drastically reduce the time spent to build the contract and the number of errors made during this process.

As regards future work, our main perspective is extending ITACA to take goal-oriented adaptation into account. Our interactive environment would accept the graphical specification of temporal logic formulas to be used next as a guide to the adaptation process. In addition, we plan to equip the toolbox with dynamic evaluation techniques as well. In such a way, the user could be dynamically informed about the satisfaction of a specified property during the construction of adaptation contracts.

Acknowledgements. This work has been partially supported by the project TIN2008-05932 funded by the Spanish Ministry of Innovation and Science (MICINN), and project P06-TIC-02250 funded by the *Junta de Andalucía*.

References

[1] ITACA's Webpage. Accesible from Javier Cámara's Webpage.

[2] M. Autili, P. Inverardi, A. Navarra, and M. Tivoli. SYNTHESIS: A Tool for Automatically Assembling Correct and Distributed Component-based Systems. In *Proc. of ICSE'07*, pages 784–787. IEEE Computer Society, 2007.

[3] A. Bracciali, A. Brogi, and C. Canal. A Formal Approach to Component Adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.

[4] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC'06*, volume 4294 of *LNCS*, pages 27–39. Springer, 2006.

[5] C. Canal, P. Poizat, and G. Salaün. Model-Based Adaptation of Behavioural Mismatching Components. *IEEE Transactions on Software Engineering*, 34(4):546–563, 2008.

[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 *Proc. of FACS'07*, volume 215 of *ENTCS*, pages 39–55. Elsevier, 2007.

[7] M. Dumas, M. Spork, and K. Wang. Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In *In Proc. of BPM'06*, volume 4102 of *LNCS*, pages 65–80. Springer, 2006.

[8] H. Foster, S. Uchitel, and J. Kramer. LTSA-WS: A Tool for Model-based Verification of Web Service Compositions and Choreography. In *Proc. of ICSE'06*, pages 771–774. ACM Press, 2006.

[9] X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. of WWW'04*, pages 621–630. ACM Press, 2004.

[10] H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of CAV'07*, *LNCS* 4590, pages 158–163. Springer, 2007.

[11] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, ISO, 1989.

[12] J. A. Martín and E. Pimentel. Automatic Generation of Adaptation Contracts. In *Proc. of FOCLASA'08*, ENTCS, 2008. To appear.

[13] 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*, LNCS, pages 84–99. Springer, 2008.

[14] R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Science of Computer Programming*, 46(3):255–281, 2003.

[15] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-Automated Adaptation of Service Interactions. In *Proc. of WWW'07*, pages 993–1002. ACM Press, 2007.

[16] M. Ouederni. Measuring Similarity of Service Protocols. Master Thesis, University of Málaga. Available on Meriem Ouederni's Webpage, Sept. 2008.

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

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