Foundational Aspects of Service Discovery based on Behavioural Contracts

Mario Bravetti
University of Bologna
bravetti@cs.unibo.it

In the context of Service Oriented Computing behavioural contracts are descriptions of the observable message-passing behavior of services. In other terms, contracts are behavioural interfaces that can be used, for instance, to check whether a group of services can be safely combined avoiding, e.g., undesired deadlocks. In this paper we consider the problem of discovering available services that can be used to implement a given service system. The idea is to first design a service system by describing the overall behaviour of each of its participant, and then instantiate such participants retrieving services exposing a behavioural contract which is conformant with the corresponding given behaviour.

Service Oriented Computing, Formal Methods, Run-time System Design

1. INTRODUCTION

Service Oriented Computing (SOC) is a paradigm for distributed computing based on services intended as autonomous and heterogeneous components that can be published and discovered via standard interface languages and publish/discovery protocols. Web Services are the most prominent service oriented technology: Web Services publish their interface expressed in WSDL, they are discovered through the UDDI protocol, and they are invoked using SOAP.

Even if one of the declared goal of Web Services is to support the automatic discovery of services, this is not yet practically achieved. Two main problems are still to be satisfactorily solved. The first one, investigated by the semantic web research community, is concerned with the lack of semantic information in the description of services. The second problem, addressed in this paper, is concerned with the problem of guaranteeing that the interacting services are compliant in the sense that their behaviours are complementary. In particular, it is important to check whether in a set of services, combined in order to collaborate, no service deadlocks waiting indefinitely for a message that never arrives.

In order to be able to check the compliance of the composed services, it is necessary that the services expose in their interface also the description of their expected behaviour. In the service oriented computing literature, this kind of information is referred to as the behavioural *service contract* [12]. More precisely, the service contract describes the sequence of input/output operations that the service intends to execute within a session of interaction with other services.

Compliance checking based on the behavioural descriptions of the composed entities has been already considered, for instance, in the context of component-based systems (see e.g. [10, 2, 19]) or for client-service interaction [11]. In this paper, we consider a different scenario with respect to both approaches.

As far as component-based systems are concerned, the commonly adopted approach is to synthesize either *wrappers* or *adaptors* that respectively block (the non compatible) part of the behaviour of one component or deal with possible mismatchings between the combined components. The approach adopted in this work is different because we address the problem of composition without the introduction of any additional wrapper or adaptor. In other terms, we consider the problem of retrieving some already available services in order to implement a correct composition without the introduction of any additional element. In the service oriented computing literature, the approach we consider is known with the name of *choreography* [23], which contrasts with the *orchestrated* approach [21] according to which all services communicate only with a central orchestrator. It is worth mentioning the fact that we could define our theory having in mind components instead of services. Nevertheless, our assumption about the choreographic approach makes all our theory more related to the current vision of service oriented computing.

As far as client-service interaction is concerned, we assume a more general context in which an arbitrary number of interacting services communicate directly without the presence of any central coordinator. We call this different context *multi-party* composition. Moving from a simpler client-service to a more complex multi-party scenario introduces several interesting new problems such as independent refinement. By independent refinement we mean the possibility to replace several services in a composition with other services that are selected one independently from the other ones.

More precisely, the aim of this work is to exploit the notion of behavioural service contracts in order to define a theory that, on the one hand, permits to formally verify whether they are compliant (thus giving rise to a correct composition) and, on the other hand, permits to replace a service with another one without affecting the correctness of the overall system. In this case we say that the initially expected contract is replaced with one of its *subcontracts*.

We intend to formalize a notion of subcontract to be exploited in the *service discovery* phase. Consider, for instance, a service system defined in terms of the behavioural contracts to be fulfilled by each of the service components. The actual services to be combined could be retrieved independently one from the other (e.g. querying contemporaneously different service registries) collecting services that either expose the expected contract, or one of its subcontracts. Another application that we foresee for our notion of subcontract is for *service updates*, as a mean to ensure backward compatibility. Consider, e.g., a service that should be updated in order to provide new functionalities; if the new version exposes a subcontract of the previous service, our theory ensures that the new service is a correct substitute for the previous one.

2. TECHNICAL CONTRIBUTION

The main contribution of this work, detailed in [9], is to generalize results that we have presented in [3, 7]. In [3, 7] we have introduced our approach for the definition of a subcontract relation: we first assume that this relation should be a pre-order, then we formalize the property that we want a "good" subcontract pre-order should preserve, and finally we define our relation as the maximum of such pre-orders (i.e. the union of all pre-orders satisfying the considered property). This is similar to the co-inductive approach considered, e.g., in the definition of the bisimulation relation for CCS [20]. More precisely, in [3, 7] we have presented a theory, developed following this approach, considering two specific languages, one for contracts and one for service systems. In this work we generalize such theory in two ways. On the one hand, we do not consider any specific contract language thus presenting a version of our theory that can be applied to any contract language satisfying a property, called *output persistence*, that we will discuss in the following. On the other hand, we do not make any specific assumption on the way service systems are specified (in [3, 7] we defined a subcontract relation assuming a precise form of service system specifications in which the restriction operator is applied directly to contracts and not to parallel compositions of contracts).

More formally, we consider a generic language for behavioural contracts essentially consisting of a process algebraic representation of labeled transition systems defined on internal, input and output actions, and an additional action representing successful completion. Then, we define a language for service system specification that simply allows for the composition of contracts with the parallel and restriction operators. We use this latter language to formalize the notion of compliance: n services/contracts are compliant if their composition is guaranteed to successfully complete without deadlocks or livelocks. After having formalized compliance, we are able to formalize the property that each refinement should satisfy: a refinement is a subcontract pre-order if it preserves service compliance, namely, given n compliant services, and substituting each of them with one of its refinements, the achieved n services are still compliant. Then we define the *subcontract relation* as the union of all subcontract pre-orders. One of the main results we have proved is that for the class of behavioural contracts that we consider, the subcontract relation achieved according to this approach is actually the largest subcontract pre-order thus allowing for the independent replacement/retrieval of contracts. In fact, in other theories of contracts recently proposed in the literature (details are reported in the next subsection), independent replacement is not allowed.

This difference with respect to other contract theories relies on the *output persistence* property that we impose on behavioural contracts: a contract is output persistent if once a contract reaches a state in which it can perform an output operation, this operation must be eventually executed from the contract before successful completion. This property is usually satisfied by languages for composing services, such as WS-BPEL [21], in which output operations cannot be guard in external choices. In these languages, once an output action is executable by a process, this output must be executed before the process successfully completes.

Another important technical achievement of [3, 7] is a characterization of the subcontract relation in a testing-like scenario [15]: we can prove that a contract C' is a subcontract of C if, after some appropriate transformations applied to both C' and C, the former is guaranteed to satisfy at least all the tests satisfied by the latter. In particular, we show how to use the theory of *should-testing* [22] to prove that one contract is a subcontract of another one. An important consequence of this characterization is a precise localization of our refinement with respect to traditional refinements such as failure refinement, or simulation (i.e. half-bisimulation): the refinement that we achieve as the largest one preserving compliance is coarser than both failure refinement and simulation.

3. RELATED WORK

As stated above, we resort to the theory of testing, in particular, to the must-testing pre-order. There are some significant differences between our form of testing and the traditional one proposed by De Nicola-Hennessy [15]. The most significant difference is that, besides requiring the success of the test, we impose also that the tested process should successfully complete its execution. This further requirement has important consequences; for instance, we do not distinguish between the always unsuccessful process 0 and other processes, such as[1] a.1 + a.b.1 for which there are no guarantees of successful completion in any possible context. Another significant difference is in the treatment of divergence: we do not follow the traditional catastrophic approach, but the fair approach introduced by the theory of should-testing of Rensink-Vogler [22]. In fact, we do not impose that all computations must succeed, but that all computations can always be extended in order to reach success.
It is well known that the De Nicola-Hennessy must testing pre-order and the CSP failure refinement [17] coincide (at least for finitely branching processes without divergences [14]). It is interesting to say that the failure refinement has been already exploited for checking component compatibility by Allen and Garlan in [1]. Similarly to our theory, the failure refinement is used to prove that a component can be replaced by one of its refinements in a component composition. Differently from our theory, a composition of several components is obtained adding a *Glue* component which behaves as a mediator for every component interaction. This Glue component permits to cut the additional actions that the refined components may include. The main difference with our theory is that, in our context, we have no mediator that allows us to cut additional behaviours of refined services. Nevertheless, the output persistence property that we consider allows us to replace a service with another one having additional behaviour.
Behavioural contracts have been initially introduced in the context of process calculi by Fournet et al. [16]. As far as service oriented computing is concerned, an initial theory of contracts for client-service interaction has been proposed by Carpineti et al. [11] and then independently extended along different directions by Bravetti and Zavattaro (see, e.g., [3-9]) by Laneve and Padovani [18], and by Castagna et al. [13]. See [9] for a detailed comparison.

REFERENCES.

[1] Allen, R., Garlan, D.: Formalizing Architectural Connection, Proc. ICSE'94, IEEE Computer Society Press, 1994, 71-80.
[2] Autili, M., Inverardi, P., Navarra, A., Tivoli, M.: SYNTHESIS: a tool for automatically assembling correct and distributed component-based systems, Proc. ICSE'07, IEEE Computer Society Press, 2007, 784-787.
[3] Bravetti, M., Zavattaro, G.: Contract based Multi-party Service Composition, Proc. FSEN'07, LNCS 4767, Springer, 2007, 207-222.
[4] Bravetti, M., Zavattaro, G.: A Theory for Strong Service Compliance, Proc. Coordination'07, LNCS 4467, Springer, 2007, 96-112.
[5] Bravetti, M., Zavattaro, G.: Towards a Unifying Theory for Choreography Conformance and Contract Compliance, Proc. SC'07, LNCS 4829, Springer, 2007, 34-50.
[6] Bravetti, M., Zavattaro, G.: Contract Compliance and Choreography Conformance in the Presence of Message Queues, Proc. WS-FM'08, LNCS 5387, Springer, 2008, 37-54.
[7] Bravetti, M., Zavattaro, G.: A Foundational Theory of Contracts for Multi-party Service Composition, Fundamenta Informaticae 89(4):451-478, IOS Press, 2008.
[8] Bravetti, M., Zavattaro, G.: A Theory of Contracts for Strong Service Compliance, Mathematical Structure in Computer Science, in publication, Cambridge University Press, 2009.
[9] Bravetti, M., Zavattaro, G.: Service Discovery based on Behavioural Contracts, in ``International School on Formal Methods for the Design of Computer, Communication and Software Systems: Web Services, SFM-09:WS, Revised Lectures'', Bertinoro, Italy, June 1-6, 2009, LNCS 5569, Springer, 2005, 261-295.
[10] Brogi, A., Canal, C., Pimentel, E.: Component adaptation through exible subservicing, Science of Computer Programming, 63, Elsevier, 2006, 39-56.
[11] Carpineti, S., Castagna, G., Laneve, C., Padovani, L.: A Formal Account of Contracts for Web Services, Proc. WS-FM'06, LNCS 4184, Springer, 2006, 148-162.
[12] Carpineti, S., Laneve, C.: A Basic Contract Language for Web Services, Proc. ESOP'06, LNCS 3924, Springer, 2006, 197-213.
[13] Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services, Proc. POPL'08, ACM Press, 2008, 261-272.
[14] De Nicola, R.: Extensional equivalences for transition systems, Acta Informatica, 24(2), Springer, 1987, 211-237.
[15] De Nicola, R., Hennessy, M.: Testing Equivalences for Processes, Theoretical Computer Science, 34, Elsevier, 1984, 83-133.

[1] We use “0” to denote unsuccessful termination, “1” for successful completion and “_+_” for choice composition.

[16] Fournet, C., Hoare, C. A. R., Rajamani, S. K., Rehof, J.: Stuck-Free Conformance, Proc. CAV'04, LNCS 3114, Springer, 2004, 242-254.
[17] Hoare, C. A. R.: Communicating Sequential Processes, Prentice-Hall, 1985.
[18] Laneve, C., Padovani, L.: The must preorder revisited - An algebraic theory for web services contracts, Proc. Concur'07, LNCS 4703, Springer, 2007, 212-225.
[19] Mateescu, R., Poizat, P., Sala un, G.: Behavioral adaptation of component compositions based on process algebra encodings, Proc. ASE'07, ACM Press, 2007, 385-388.
[20] Milner, R.: Communication and Concurrency, Prentice-Hall, 1989.
[21] OASIS: WS-BPEL: Web Services Business Process Execution Language Version 2.0, Technical Report, OASIS, 2003.
[22] Rensink, A., Vogler, W.: Fair testing, Information and Computation, 205, Elsevier, 2007, 125-198.
[23] W3C: WS-CDL: Web Services Choreography Description Language, Technical Report, W3C, 2004.