Implicit vs. Explicit Data-Flow Requirements in Web Service Composition Goals

Annapaola Marconi, Marco Pistore, and Paolo Traverso

ITC-irst
Via Sommarive 18, Trento, Italy
{marconi, pistore, traverso}@itc.it

Abstract. In this paper we compare two different approaches to specify data-flow requirements in Web service composition problems, i.e., requirements on data that are exchanged among component services. *Implicit* data-flow requirements are a set of rules that specify how the functions computed by the component services are to be combined by the composite service. They implicitly define the required constraints among exchanged data. *Explicit* data-flow requirements are a set of explicit specifications on how the composition should manipulate messages and route them from/to components. In the paper, we compare these two approaches through an experimental evaluation, both from the point of view of efficiency and scalability and from that of practical usability.

1 Introduction

Service composition is one of the fundamental ideas underlying service-oriented applications: composed services perform new functionalities by interacting with component services that are available on the Web. In most real-world applications, service compositions must be at the "process-level", i.e., they must take into account that component services are stateful processes [1] and that they require to follow complex interaction protocols specified in language such as WS-BPEL [2], The automated synthesis of composed services is one of the key tasks that supports the design and development of service oriented applications: given a set of available component services and a composition goal, the task corresponds to the synthesis of a composition, e.g., a new service, or a set of constraints on the behaviors of existing services, which satisfies the requirements expressed by the composition goal.

Recent works address the problem of the automated synthesis of composed services at the process level, see, e.g., [1,3,4,5,6,7]. However, most of them do not take into account a key aspect of the composition problem: the specification of *data-flow requirements*, i.e. requirements on data that are exchanged among component services. This is a significant and challenging problem, since, in real life scenarios, business analysts and developers need a way to express complex requirements on the exchanged data. Moreover, to make the automated composition an effective and practical task, the requirements specification should be easy to write and to understand for the analyst. Surprisingly, very little effort has been devoted in the literature to address this problem.

In this paper we compare two different approaches to the specification of data-flow requirements for the automated synthesis of composed services specified in WS-BPEL.

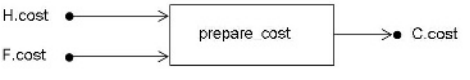
A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 459–464, 2006.

The first approach is based on *implicit data-flow requirements* [7]. It exploit the functions that define the tasks carried out by the component services, and that annotate their WS-BPEL descriptions. Composition goals contain references to such functions that implicitly define constraints on data flows. Consider, for instance, a virtual travel agency (VTA from now on) that composes two component services, a Flight and a Hotel reservation service. The implicit data-flow requirement stating that the cost offered to the Customer is a specific function (prepare_cost) of the costs of the Hotel and of the Flight can be specified as follows:

$$\text{C.cost} = \text{prepare\_cost}(\text{H.costOf}(\text{C.loc},\text{C.date}),\text{F.costOf}(\text{C.loc},\text{C.date}))$$

We assume that the cost functions F.costOf and H.costOf appear as "semantic" annotations in the WS-BPEL processes of the flight and hotel components, respectively. This requirement implicitly specifies a data flow from the messages of the components to the composed service. The composition task should determine from this specification which messages and which data should be sent from a service to another. In [7] it is shown how the framework for process-level composition defined in [5,6] can be exploited to generate the composition starting from implicit data-flow requirements.

The second approach is based on *explicit data-flow requirements*. In this case, composition goals contain constraints that define the valid routings and manipulations on the messages of the component services, i.e., these constraints specify how the output messages of the composed service are obtained by manipulating and combining in suitable ways the input messages obtained from the component services. In the VTA example, the explicit data flow requirement is the following:



It directly specifies that the VTA must apply its internal function prepare_cost on the costs received from the Hotel and from the Flight, to obtain the cost to be sent in the offer to the Customer. In [8] it is shown that explicit data-flow requirements can be described in a graphical way, as *data-nets*. It is also shown how to adapt the composition framework of [5,6] to the case of explicit data-flow requirements.

2 Comparison of the Two Approaches

We now compare the two proposed approaches for modeling data-flow requirements from a technical point of view, from the point of view of the performance, and for what concerns usability.

The two approaches present some similarities. First of all, both of them extend the automated composition framework proposed in [6,5]. Moreover, the analysis of the approaches described in [7] and in [8] shows that they adopt the same strategy to encode data manipulation and exchange at an abstract level in the composition domain: introducing goal variables (which model variables of the new composite process) and encoding data-flow requirements as constraints on the operations that can be performed

on goal variables. Another similarity is that both approaches do not require to enumerate the values that these variables can assume; this task would be impossible since variables often require very large or infinite ranges that prevent an explicit enumeration — consider for instance the variables representing the costs in the VTA example. The main difference lies in the way the two approaches reason on goal variables. The implicit approach reasons on what is "known" about the goal variables in the states of the composition domain (e.g., H.costOf(C.loc, C.date) becomes a goal variable, and the composition approach reasons on whether the value of this variable is "known"), and uses this information to check whether the goal is satisfied. The explicit approach does not encode at all data within the composition domain: its states simply model the evolution of the processes, and data-flow constraints are modeled as additional "services" (e.g. the requirement in the VTA example is modeled as a 'service' that transforms the flight and hotel cost into the cost for the customer). This difference is reflected in the size of the obtained composition domain, which in the implicit approach is much larger.

In order to test the performance of the proposed approaches, we have conducted some experiments on a scalable domain. Since we wanted to compare the two approaches on realistic domains, we consider here a real e-commerce scenario, the Virtual Online Shop (VOS from now on). The VOS consists in providing an electronic purchase and payment service by combining a set of independent existing services: a given number of e-commerce services Shops and a credit-card payment service Bank. This way, the Customer, also described as a service, may directly ask the composite service VOS to purchase some given goods, which are offered by different Shops, and pay them via credit-card in a single payment transaction with the Bank. For the Bank we modeled a real on-line payment procedure offered by an Italian bank. Such a process handles several possible failures: it checks both the validity of the target bank account (the Store's one in our case) and the validity of the credit card, it checks whether the source bank account has enough money and whether the owner is authorized for such a money transfer. The Shop models a hypothetical e-commerce service, providing a complex offer negotiation and supporting a transactional payment procedure. This composition problem requires a high degree of interleaving between components (to achieve the goal, it is necessary to carry out interactions with all component services in an interleaved way) and both the implicit and explicit models of data-flow requirements are pretty complex (due to the number of functions and to the need of manipulating data in a complex way). To evaluate the scalability of the two approaches when the number of (complex) component services grows, we increased the number of stores participating to the composition. The following table reports the experimental results.

	Implicit					**Explicit**						**WS-BPEL**
	domain			**time (sec.)**		**domain**				**time (sec.)**		num
	goal vars	nr. of states	max path	model construction	composition & emission	goal vars	data constr.	nr. of states	max path	model construction	composition & emission	complex activities
VOS	6	1357	54	11.937	3.812	14	6	390	26	1.612	0.218	52
VO2S	9	8573	64	185.391	84.408	20	9	1762	32	1.796	0.688	84
VO3S	12	75289	74	M.O.	-	26	12	12412	38	1.921	2.593	109
VO4S	-	-	-	-	-	32	15	122770	44	2.218	12.500	136
VO5S	-	-	-	-	-	38	18	1394740	50	2.547	26.197	165
VO6S	-	-	-	-	-	44	21	16501402	56	2.672	246.937	196

For each considered scenario the table shows some parameters that characterize the complexity of the composition domain, the automated composition time, and the size of the generated composite process. The complexity of the implicit approach is given in terms of the number of goal variables that encode the pieces of "knowledge" that the composite process acquires while interacting with the component services and manipulating messages (see [7] for the details). For what concerns the explicit approach, we consider the number of data-flow constraints and the number of goal variables (as shown in [8], the variables correspond to number of nodes of the data net obtained by combining all the data-flow constraints). To complete the characterization of the complexity of the composition domain, for both approaches we report the number of states and the number of transitions of the longest path in the composition domain that is passed as input to the automated generation techniques of [5,6]. These measures characterize the size of the search space for the composed service.

The complexity of the composition task can also be deduced from the size of the new composite WS-BPEL process, which is reported in the last column of the table. We remark that we report the number of WS-BPEL basic activities (e.g. `invoke`, `receive`, `reply`, `assign`, `onMessage`) and do not count the WS-BPEL structured activities that are used to aggregate basic activities (e.g. `sequence`, `switch`, `flow`). Indeed, the former activities are a better measure of the complexity of the generated process, while the latter are more dependent on the coding style used in the composite WS-BPEL process. Notice that we report only one measure for the composite process. Indeed, the processes generated by the two approaches are basically identical: they implement the same strategy, handle exceptions and failures in the same way and present the same number of activities. The only difference is the way in which such activities are arranged, e.g. the order of invocation of the different shops or of the assignments when preparing different parts of a message to be sent.

The composition times have been obtained on a Pentium Centrino 1.6 GHz with 512 Mb RAM of memory running Linux. We distinguish between model construction time and composition and emission time. The former is the time required to obtain the composition domain, i.e., to translate the WS-BPEL component services into a finite state domain and to encode the composition goal. The latter is the time required to synthesize the controller according to [5,6] and to emit the corresponding WS-BPEL process. The experiments show that the implicit approach has worse performances both for the model construction time and for the composition time. In particular, the implicit approach is not able to synthesize the VOS scenario with three shops: a memory out is obtained in model construction time. In the case of the explicit approach, instead, the time required to generate the composition domain is very low for all the scenarios, and also the performance for the composition scales up to very complex composition scenarios: the VO6S example (6 Stores, 1 Bank and 1 Customer) can be synthesized in about 4 minutes. We remark that this example is very complex, and requires several hours of work to be manually encoded: the corresponding WS-BPEL process contains about 200 non-trivial activities! We also remark that the number of states in the implicit domain for the VO6S example is much larger than the number of states of the VO3S example in the explicit approach. The fact that the former composition has success while the latter has not shows another important advantage of the explicit approach: the domain

is very modular, since each data-flow constraint is modeled as a separated "service", which allows for a very efficient exploitation of the techniques implemented by [5,7].

For what concerns usability, the judgment is not so straightforward, since the two approaches adopt very different perspectives. From the one side, modeling data-flow composition requirements through a data net requires to explicitly link all the messages received from component services with messages sent to component services. Moreover, a second disadvantage is that component services are black boxes exporting only input and output ports: there is no way to reason about their internal data manipulation behaviors. From the other side, data nets models are easy to formulate and understand through a very intuitive graphical representation. It is rather intuitive for the designer to check the correctness of the requirements on data routings and manipulations. Also detecting missing requirements is simple, since they usually correspond to WS-BPEL messages (or part of messages) that are not linked to the data net. Finally, the practical experience with the examples of the experimental evaluations reported in this section is than the time required to specify the data net is acceptable, and much smaller that the time required to implement the composite service by hand. In the case of the VO6S scenario, for instance, just around 20 minutes are sufficient to write the requirement, while several hours are necessary to implement the composite service.

The implicit approach adopts a more abstract perspective, through the use of annotations in the process-level descriptions of component services. This makes the goal independent from the specific structure of the WS-BPEL processes implementing the component services, allowing to leave out most implementation details. It is therefore less time consuming, more concise, more re-usable than data nets. Moreover, annotations provide a way to give semantics to WS-BPEL descriptions of component services, along the lines described in [9], thus opening up the way to reason on semantic annotations capturing the component service internal behavior. Finally, implicit knowledge level specifications allow for a clear separation of the components annotations from the composition goal, and thus for a clear separation of the task of the designers of the components from that of the designer of the composition, a separation that can be important in, e.g., cross-organizational application domains. However, taking full advantage of the implicit approach is not obvious, especially for people without a deep know-how of the exploited composition techniques. There are two main opposite risks for the analysts: to over-specify the requirements adding non mandatory details and thus performing the same amount of work required by the explicit approach; to forget data-flow requirements which are necessary to find the desired composite service. Moreover, our experiments have shown that, while the time required to write the implicit data-flow requirements *given* the annotated WS-BPEL processes is much less than the time to specify the data net, the time required to write the requirements *and* to annotate the WS-BPEL processes is much more (and the errors are much more frequent) than for the data net.

3 Conclusions and Related Works

We have compared two different approaches to the definition of data flow requirements for the automated synthesis of process-level compositions for component services described in WS-BPEL. Implicit requirements [7] compose functions that annotate

WS-BPEL descriptions of component services. Explicit requirements [8] directly specify the routing and manipulations of data exchanged in the composition. Both approaches have their pros and cons. Explicit models allow for much better performances, due to the very modular encoding of requirements. Moreover, they are rather easy to write and to understand for the analyst. Implicit requirements allow for a higher degree of re-use and abstraction, as well as for a clear separation of the components annotations from the composition goal.

Most of the works that address the problem of the automated synthesis of process-level compositions do not take into account data flow specifications. This is the case of the work on synthesis based on automata theory that is proposed in [1,3], and of work within the semantic web community, see, e.g., [10]. Some other approaches, see, e.g., [11], are limited to simple composition problems, where component services are either atomic and/or deterministic.

The work closest to ours is the one described in [4], which proposes an approach to service aggregation that takes into account data flow requirements. The main difference is that data flow requirements in [4] are much simpler and at a lower level than in our framework, since they express direct identity routings of data among processes, and do not allow for manipulations of data. The examples reported in this paper clearly show the need for expressing manipulations in data-flow requirements and higher level requirements.

References

1. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-Services: A Look Behind the Curtain. In: Proc. PODS'03. (2003)
2. Andrews, T., Curbera, F., Dolakia, H., Goland, J., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weeravarana, S.: Business Process Execution Language for Web Services (version 1.1) (2003)
3. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of Services with Nondeterministic Observable Behaviour. In: Proc. ICSOC'05. (2005)
4. Brogi, A., Popescu, R.: Towards Semi-automated Workflow-Based Aggregation of Web Services. In: Proc. ICSOC'05. (2005)
5. Pistore, M., Traverso, P., Bertoli, P., A.Marconi: Automated Synthesis of Composite BPEL4WS Web Services. In: Proc. ICWS'05. (2005)
6. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: Proc. ICAPS'05. (2005)
7. Pistore, M., Marconi, A., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: Proc. IJCAI'05. (2005)
8. Marconi, A., Pistore, M., Traverso, P.: Specifying Data-Flow Requirements for the Automated Composition of Web Services. In: Proc. SEFM'06. (2006)
9. Pistore, M., Spalazzi, L., Traverso, P.: A Minimalist Approach to Semantic Annotations of Web Processes. In: Proc. of ESWC'05. (2005)
10. McIlraith, S., Son, S.: Adapting Golog for Composition of Semantic Web Services. In: Proc. KR'02. (2002)
11. Ponnekanti, S., Fox, A.: SWORD: A Developer Toolkit for Web Service Composition. In: Proc. WWW'02. (2002)