Compositional logical semantics for business process languages

Mihhail Matskin
Dept. of Electronics, Computer and Software Systems
Royal Institute of Technology - KTH
Stockholm, Sweden
misha@imit.kth.se

Riina Maigre, Enn Tyugu
Institute of Cybernetics at TUT
Tallinn, Estonia
{riina,tyugu}@cs.ioc.ee

Abstract—In this paper we propose a compositional logical semantics for business process languages to be used in automatic Web service composition. We introduce a concept of Higher Order Work Flow (HOWF) and use it for expressing control of business process. A precise semantics of HOWF enables us both to dynamically generate HOWF for automatic composition of services and to reason about the reachability of goals in process models when HOWF are described manually. Our semantics is general enough to cover different process languages; however, we mainly show its applicability in the context of OWL-S and BPEL.

Keywords—Web service composition; business process language, logical compositional semantics

I. INTRODUCTION

Service composition is a hot research topic in the area of Web services [1,2]. It considers writing a composite process where component processes are other Web services. Web service composition is sometimes interchanged with another term - Web service orchestration. However, we consider composition when we refer to the whole process and models of Web service composition, and orchestration when we define models and languages used to describe the order in which services are invoked. We prefer to have such separation because we would like to consider presentation of the composite services separately from the process of generation of such presentation.

In order to support presentation of composite services one can use different orchestration models, some examples of these models are activity hierarchies, statecharts, activity diagrams, Petri-nets and rules [3]. Orchestration models are implemented in many business process languages [4,5]. In particular, a well-known orchestration language BPEL [4] can be viewed as having activity hierarchies model. The choice of a particular orchestration model may depend on desirable characteristics of the presentation – expressiveness, efficiency of compilation or possibility to dynamically generate composite Web service.

While the orchestration models are now quite well developed, most of these models are focused on executional/computational aspects of services [6,7,8,9] and do not have precise semantics that helps dynamic synthesis of composite services.

In this paper we propose an abstract orchestration model that is based on Higher Order Flows (HOWF). This model hides details of Web services that are not essential for dynamic composition into realization and, at the same time, it allows instantiation of the composite service with constructs of a particular process language. In other words, we do not provide a precise semantics of practical process languages (because our logical representation is not expressive enough for that), but we give an abstract representation of a practical orchestration language, provide its precise semantic in logic and use formal logical system for automatic generation of a composite Web service as a workflow. When a workflow is created, we instantiate it with constructs from practical particular business process language. In other words, we are going to talk about Web service composition from a perspective of presentation of composite service workflow and compositional semantics of such workflow.

It is important to mention that this work is based on our long-term activity in automatic program synthesis and it adopts the developed synthesis methods and techniques [10] to Web services. Software synthesis is a tough problem. Web service composition is also not a simple work, but because of nice structural properties of Web services (loose coupling and simple control structures in many cases) we hope that the composition methods might be easier to apply in practice.

This paper is organized as follows. First we consider workflows for automatic service composition and discuss how to represent Web services (tasks) and control constructs in logic and, particularly, in intuitionistic logic. Based on that, we consider a general schema for Web service composition as well as generation of composite services process description in a business process language. Finally, we give an example of implementation of this approach by using an existing tool.

II. HIGHER ORDER WORKFLOW

In this and in the following section we present an orchestration model based on HOWF and a logic that enables us both to generate higher-order workflow (automatic composition of service) and to reason about the reachability of goals on workflow models. This is a suitable basis for dynamic composition of services. The approach extends workflow semantics by enabling specification of complex algorithms in terms of workflow. We are going to represent the logic on an
example and try to avoid purely syntactic formal representation as much as possible.

It is easy to represent partial order of services by a graph. However, the problem is how to express control in this graph. Various fixed patterns of choice can be utilized for this. In particular, Petri nets are used for this purpose [9]. However, to achieve a general solution we might need higher order features in the representation formalism, because the objects to be manipulated are the services themselves.

Let us consider an example of a travel planning service. We assume that this is a composite service that supports students in planning a trip around Europe during their summer vacations. We are going to use the existing atomic services chooseLeg and addLeg for making a choice of a new travel leg and adding it to the itinerary. There also has to be a payment service (finish), e.g. giving the credit card number and authorizing the purchase. In business process execution languages [4], we have control constructs sequence, cycle and other that vary from language to language to some extent, but can all be used in our case. Because the services chooseLeg and addLeg must be repeated several times (the number of repetitions is not known beforehand), our composite service should introduce a control over services that forms a cycle. We can represent the workflow very abstractly as a graph shown in Fig. 1. In this case we have two types of services in a composite service workflow: basic and control or structural. Basic services (white ovals) are atomic services (for example, travel service for planning one leg of the trip) and control services (blue ovals) are loops, choice, sequences or other possible control patterns over other services. Thin lines in Fig. 1 express the ordering of the invocation of services and thick lines express control that cycle and sequence have over the other services. It is important to mention that types of control services with incoming thick lines are one order higher than other services at the outgoing end of a thick line, because they have these other services (both basic and control) as parameters.

The data dependencies are shown in Fig. 2, where the data entities: departure, itinerary, etc. are explicitly shown together with their roles as inputs and outputs of services. We have introduced the data entities departure, leg, booked, departure1 (denoting the intermediate departure point of the travel), itinerary and confirmation. Their meaning should be rather obvious. The extension by data dependencies make some control information implicit – order of services may be defined already by data dependencies. When the order is determined by data dependencies, one can drop the explicit ordering control information. This is what we are going to use in composition process – derive control information from data dependencies. A particular type of control node is defined by the label of the control node of HOWF and can be utilized for choosing of appropriate construct of practical business process language.

We are interested in providing a precise semantics to our HOWF, in particular, to define its logical semantics. Representation in logic has the advantage that it allows to reason in a very precise way about the workflow. It enables us both to generate HOWF (automatic composition of processes) and to reason about the reachability of goals in workflow models. General relations between logical presentation, HOWF and process languages are shown in Fig. 3.
(and vice versa) that allows one to synthesize a new HOWF from a proof. At the same time, HOWF can be transformed into business process languages via instantiating control nodes by the language constructs, which allows one to generate composite processes (composite Web services) automatically. In this case we can say that HOWF represent a composite semantics of process languages.

Several equivalent formalisms for representing higher order dependencies have been considered that cover both data dependencies and control: higher-order functional constraints (HOFCN) [12], higher-order dataflow schemas [13] and a fragment of intuitionistic logic [11]. We are going to use logic for defining the semantics of workflow. Representation in logic has the advantage that it allows to reason in a very precise way about the workflow.

III. REPRESENTING HOWF IN LOGIC

As long as the functionality of basic services of a workflow is expressed only on the level of computability of outputs, one can use intuitionistic propositional logic for representing the semantics of workflow. For example, the functionality of services chooseLeg and addLeg is representable by the following implications where the data names have the role of propositions:

\[
\text{departure1} \Rightarrow \text{leg}
\]

\[
\text{leg} \Rightarrow \text{booked}
\]

The first tells us that having departure1 one can obtain leg. The second tells us that having leg one can obtain booked.

According to the standard semantics of intuitionistic logic the meaning of a proposition is not a truth-value. It is an object of a proper type [14], and in our case it will always be the data item that is the respective input or output of the service. According to the semantics of intuitionistic logic, the implications have a computational meaning, or in other words – their realizations must be functions. In our case they are the services chooseLeg, bookLeg and finish. They can be shown in logical style as the realizations of the formulas written after colon:

\[
\text{departure1} \Rightarrow \text{leg} : \text{chooseLeg}
\]

\[
\text{leg} \Rightarrow \text{booked} : \text{bookLeg}
\]

\[
\text{itinerary} \Rightarrow \text{confirmation} : \text{finish}
\]

Now we are going to the representing of control. The difference between a basic service and a control node is that the control node, in addition to simple data inputs and outputs, has other services as parameters. However, we do not want to operate in logic with names of services (this would require the usage of the higher order logic), but with logical propositions only. Therefore we describe the services that are parameters by means of their pre- and postconditions, binding them with control nodes. This is denoted by dashed arrows in Fig. 4. In this figure, c1 and c2 are ordinary data input and output, while c8 and c9 express pre- and postcondition of a node (or of a sequence) that is a parameter to the cycle.

![Figure 4. A control node](image)

Logically this is represented by a nested implication. In the present case it gives the formula

\[
c1 \land (c8 \Rightarrow c9) \Rightarrow c2 : \text{cycle}.
\]

Now we can change the notation and use no thick arrows at all, but only thin (normal or dashed) arrows. This allows us to use uniform representation and binding data for all workflow models as shown in Fig. 5 for the travel example.

![Figure 5. Expressing higher order workflow with data dependencies](image)

The logic behind this workflow is expressed by four formulas, one formula for every service and control node in the workflow. Let us look now at the control node cycle. It has a precondition departure and a postcondition itinerary. Besides that, it produces a precondition departure1 for the sequence of services that constitute the body of the cycle. This precondition expresses the availability of current departure (for the current leg of the trip) and requires a postcondition booked of the sequence. The postcondition booked expresses the availability of a partial itinerary that has the newly found leg as its last part. All this is expressed in the logic as follows:

\[
departure \land (\text{departure1} \Rightarrow \text{booked}) \Rightarrow \text{itinerary} : \text{cycle}.
\]
Indeed, the implication \( \text{departure} \supset \text{booked} \) represents the functional input realized by a sequence of services, hence it must be on the input side of the main implication for \( \text{cycle} \). It is important to understand that the implication \( \text{departure} \supset \text{booked} \) represents an input of the control node \( \text{cycle} \), but its value is not a data! It is body of the loop performed by the control node \( \text{cycle} \). Here the higher order is hidden now. This body must be synthesized from other available services, and we call such implication on the left side of another implication a subtask.

All other nodes are described by simple implications:

\[
\text{departure} \supset \text{leg} : \text{chooseLeg} \\
\text{leg} \supset \text{booked} : \text{bookLeg} \\
\text{itinerary} \supset \text{confirmation} : \text{finish}
\]

The fragment of logic used here is already sufficient for representing dataflow and also synthesizing Web services as nested sequences of services taking into account their data dependencies. We postpone the discussion of derivation of sequences of services until we have introduced also proof rules of the logic.

IV. FINDING A PROOF

Not going into details, we show now the correct admissible rules for the fragment of intuitionistic logic that is used for expressing HOWF. The rules are sufficient for deriving proofs of solvability, i.e. of the derivability of a goal on a given workflow expressed in logic (see [10]). An admissible rule corresponds to a fragment of derivation with standard rules of logic, hence derivation with admissible rules is shorter than a standard derivation. There are two rules: one for services and another for control nodes. Application of a rule corresponds to invocation of a services or applying a control node. The rules are as follows (we present here the rule SSP1 for brevity with only one nested implication, i.e. with only one subtask, but this is not a principal restriction on the logic):

\[
(A \supset B) \land X \supset Z : f \quad A \land W \supset B : g \\
\frac{}{X \land W \supset Z : f(g)} \quad \text{SSP1}
\]

\[
A \supset B \land C : f \\
\frac{}{B \land D \supset G : g} \quad \text{SSP2}
\]

The letters \( A, B, C, D, G, W, Z \) are metavariables that denote propositions (data names in our case) or their conjunctions. We have omitted some structural rules here that are applicable and needed as well. The derivation rules show us also how realizations of consequents are built from realizations of premises of the rules: the rule SSP2 composes a sequence of services \( f;g \), and the rule SSP1 takes a body of loop \( g \) and gives it as an argument to the realization \( f \) of the control node. This body must be synthesized in advance as realization of the formula \( B \land D \supset G \).

The rules are sufficient for composing the correct composite service for computing \( \text{confirmation} \) from given \( \text{departure} \). This can be demonstrated by deriving in intuitionistic logic the goal of computations expressed by the implication

\[
\text{departure} \supset \text{confirmation} .
\]

This goal can be derived from the formulas that represent our three atomic services \( \text{chooseLeg}, \text{bookLeg}, \text{finish} \) and the control node \( \text{cycle} \). This derivation is short, however, the search may be quite time consuming in general, because it is known that the problem of proof search in intuitionistic propositional logic is decidable, but it is P-space complete, and our fragment of intuitionistic logic is complete in the sense of derivability [11]. However, a good strategy exists for proof search in the case of formulae restricted to implications as we have them for representing services and control nodes. This strategy combines forward search over services and backward search over control nodes. The trick is that we use admissible rules for derivation that work better than conventional inference rules of intuitionistic logic.

\[
\frac{}{\text{departure} \supset \text{confirmation} : \text{payment}} \quad \text{SSP2}
\]

\[
\text{departure} \supset \text{confirmation} : \text{payment} \\
\text{cycle} (\text{chooseLeg}; \text{bookLeg}); \text{finish}
\]

Figure 6. Derivation of a goal

Now we can build the derivation of the goal \( \text{departure} \supset \text{confirmation} \) as shown in Fig. 6. This demonstrates how the logic is used for finding the order of invocation of services together with control over their execution. The composition of services is shown in Courier font after colon at consequents of each derivation step. The complete description of the execution order appears at the last step. Pay attention to the procedural parameter \( \text{chooseLeg}; \text{bookLeg} \) of the control node \( \text{cycle} \) – this is the body of the loop synthesized automatically from two services.
The derivation of the goal from the previous section can be considered as a composition of a Web service with input departure, output confirmation and the body represented by the following order of (nested) invocations of atomic services:

\[
\text{cycle(chooseLeg;BookLeg); finish}
\]

Taking into account the dependencies shown in Fig. 3, we can present the general schema of the composition process as shown in Fig. 7.

We have instantiated the control nodes with BPEL activities in Fig. 8. Transformation of this HOWF into BPEL is quite straightforward. We can have different generators from HOWF to different process languages. Next section shows our experiments with generation of OWL-S descriptions.

VI. EXPERIMENT

There are several software tools that use intuitionistic propositional logic for automatic program construction [15]. We have used the tool CoCoViLa [15] that supports the development of visual languages for specifying software, and includes an algorithm synthesis part that is able to handle the specifications in the intuitionistic propositional logic. We have developed and implemented a visual language for representing Web services, control nodes and their compositions in the form of HOWF. A visual representation in CoCoViLa of a composite service is automatically translated into a Java program. Running the program produces a specification of the composite service in OWL-S. The result can be generated also in BPEL or as a runnable Java code that calls the services.
Fig. 9 shows the visual specification of booking a trip as we have described it in the example. The menu bar has buttons for components (visual classes) of the visual language like map (M), cycle (C) etc. An important component of the visual language is a generic service for expressing concrete services. It is used for expressing the services chooseLeg, bookLeg and finish in Fig. 9. This visual class has a pop-up window for specifying properties of a service. The figure shows the pop-up window for the service chooseLeg. The figure also shows a window where we see a part of the text in OWL-S produced as a result of the synthesis.

VII. CONCLUSIONS

We have proposed a compositional semantics of business process languages to be used in automated Web service composition. The core of our proposal is HOWF that can be considered as a generalization of business process language orchestration models. HOWF allows one to present different control patterns as higher-order control nodes and this allows one to represent processes of arbitrary complexity. The control nodes can be easily instantiated by constructs from a particular process language when generating the text in this language. Different process language generators can be used for different languages.

We have also shown that HOWF has a precise logical semantics. Representation in logic makes it possible to reason in a very precise way about the composite Web service process workflow. In particular, it enables us both to dynamically generate HOWF for automatic composition of services and to reason about the reachability of goals in process models when HOWF are described manually.

We have a system that implements HOWF and their dynamic composition as well as the tools supporting the visual presentation of HOWF manually. The experience in using the tools has convinced us in the feasibility and usefulness of the approach. At the same time we recognize that fully automatic generation of Web services in process language from description of atomic services should be enhanced by possibility to adjust and to revise the generated process flow by the customer. We are going to work on including this feature as well as more advanced debugging means into our tools in the future.

ACKNOWLEDGMENT

This work is partially supported by the Norwegian research Foundation in the framework of VERDIKT program – the FABULA project and by the Estonian Science Foundation Grant No. 6886.

REFERENCES