

An Iterative Approach for the Process-level Composition of Web Services.

Annapaola Marconi, Marco Pistore, and Paolo Traverso

FBK,
via Sommarive 18, Trento, Italy
{marconi,pistore,traverso}@fbk.eu

Abstract. Web service composition is one of the most promising and challenging ideas underlying service oriented applications, particularly in case that the components are themselves structured business processes specified in languages such as WS-BPEL. The manual development of the new composite service is a time consuming and error prone task since it must be driven by the analysis of the interaction protocols of the component services. The automated generation of the composition, on the other side, requires the specification of complex requirements that have to capture all the relevant aspects of the composition. In this paper we propose an iterative semi-automated development process for web service composition which dramatically reduces the effort for the composition task. The proposed approach automatically generates both the internal executable process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL). The composition process starts from composition requirements that can be easily specified through an intuitive graphical notation, and that can be incrementally refined on the basis of the results of the automated composition.

1 Introduction

Developing composite processes interacting with complex real world web services requires a time consuming analysis of the component services, both for what concerns their interaction protocol and the data structure of their messages. Moreover, it requires a detailed implementation of the new composite service that takes into account all the possible interaction evolutions (faults, exceptions).

The ability to automatically compose web services is an essential step to substantially decrease time and costs in the development, integration, and maintenance of web services. To be used in practice, automated composition should handle the complexity of real world component web services and being able to generate an executable, ready to run and possibly easy refining new web process. This ability is particularly relevant in those composition scenarios where component services are stateful processes [1] that require to follow complex interaction protocols specified in languages such as WS-BPEL [2].

Several works address the problem of the composition of stateful services, see, e.g., [1, 3–5]. However, the key problem of the practical applicability of these approaches in real composition scenarios is still open. Addressing this problem requires to answer questions such as how to specify the stateful behavior of the component services, how to specify the business requirements that define the goal of the composition, and whether the composition techniques are powerful enough to scale to scenarios of realistic size.

In [6] we've shown that the features offered by our composition framework - in particular the possibility to define complex, structured business requirements for the composition ([7–9]) and the very efficient underlying composition techniques ([10]) - are adequate for handling real world composition problems, such as the generation of an e-Bookstore composite service interacting with Amazon E-Commerce Services and the e-payment service offered by Banks of Monte dei Paschi di Siena.

The experiments with the e-Bookstore scenario considered in [6] highlight the need of extending our composition techniques in order to improve their applicability.

In this paper we propose an iterative development process that dramatically reduces the effort for the composition by automatically generating both the internal executable composite process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL). The approach, extending the sophisticated techniques described in [7, 10, 8], is able to handle real world composition problems.

The proposed composition process consists of two phases. The aim of the first phase is to obtain a preliminary version of the composite process starting from initial composition requirements. During this phase the developer analyses the component service protocols (abstract WS-BPEL and WSDL) and specifies control flow and data flow requirements through an intuitive graphical notation. Given the description of the component services and the requirements specification we automatically generate the internal executable composite process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL).

This preliminary version of the composite service can be iteratively enhanced in the second phase of the process. During this phase the developer, on the basis of the automated composition outcomes, can refine both the composition requirements and the customer interface and automatically re-compose.

The rest of the paper is structured as follows. In Section 2 we give an overview of the proposed approach and describe a composition scenario that we will use as a reference example. Then we describe in details each step of the proposed composition process: from control flow and data flow composition requirements specification (Section 3 and Section 4 respectively), to automated composition (Section 5) and requirements refinement (Section 6). Finally, Section 7 concludes the paper with some concluding remarks and a discussion on future work.

2 An overview of the proposed composition process

In this Section we give an overview of the proposed approach, present the different phases of the development process and briefly describe a composition scenario that we will use as a reference example in the rest of the paper.

The main idea underlying web service composition is the definition of new services (composite service) that provide new functionalities by interacting with pre-existing services (component services). We assume that component services are described using standard languages. In particular, we require that each component service is specified through its abstract WS-BPEL, describing the interaction protocol, and its WSDL, specifying all interaction details (data types, messages, operations, bindings, etc.). The aim of the proposed composition process is the development of a ready to run new composite process published as a web service. Thus the outcome of the composition is the composite service internal implementation (executable WS-BPEL) and its external interface (abstract WS-BPEL and WSDL).

The composition scenario, that we use as a reference example in the paper, is the Purchase and Ship (P&S) case study.

Example 1 (The P&S composition scenario).

The P&S example consists in providing a furniture purchase & ship service by combining two independent existing services, a furniture producer **Producer** and a delivery service **Shipper**. This way, a potential customer, may directly ask the composite service **P&S** to purchase a given item and deliver it at a given place (for simplicity, we assume that the shipment origin is fixed and leave it implicit).

The interactions with the existing services have to follow specific protocols. We assume that the component protocols are published through their WSDL and abstract WS-BPEL specifications. For instance, the interactions with the **Shipper** (see Figure 3¹) start with a request for transporting a product of a given size to a given location. This might not be possible, in which case the requester is notified through an error message. Otherwise, a delivery cost is computed and sent back to the requester. Then the **Shipper** waits for either an acceptance or a refusal of the offer from the invoker. Similarly the **Producer** protocol (see Figure 3) starts with a request for ordering a given quantity of a specific item. If the order can be fulfilled, the **Producer** sends back an offer message carrying the information about the total cost and the size of the package. Otherwise, the requester is notified through a message carrying the unavailability details. Then the **Producer** waits for either an acceptance or a refusal of the offer from the requester.

The P&S high level goal is to sell home-delivered furniture interacting with the **Shipper** and **Producer** services according to their protocols.

The composition approach we propose consists of two phases. The aim of the first phase is to obtain a first version of the composite process starting from initial composition requirements. This preliminary version is iteratively enhanced in the second phase of the process.

Figure 1 presents an high level view of the proposed composition process, while Figure 2 presents a schematic description of each step of the proposed process.

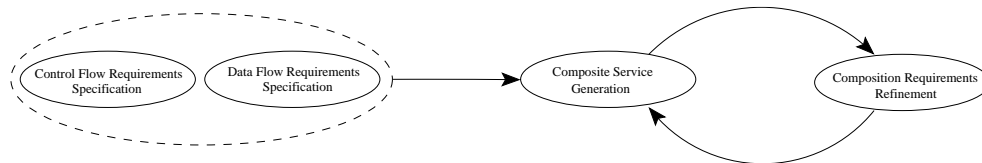


Fig. 1. The Proposed Web Service Composition Process

During the first phase, we require to specify composition requirements through an intuitive graphical notation. The approach clearly separates control flow from data flow requirement specification (as described in [8]).

In the control flow requirement specification step the developer defines termination conditions and transactional issues by simply annotating the component service abstract WS-BPEL with semantic information. If we consider P&S case study, we have that the main goal of the

¹ In all figures, labels of input transitions start with a "?", labels of output transitions start with a "!", other transitions correspond to internal operations performed by the services.

Control Flow requirements specification	
<i>Objective:</i>	Define the control flow requirements for the composition problem.
<i>Automated VS Manual:</i>	Semi-automated.
<i>Input:</i>	Component service abstract WS-BPEL.
<i>Output:</i>	Component service annotated abstract WS-BPEL, control flow requirements.
<i>Description:</i>	Semantic annotation of the component abstract WS-BPEL (manual), specification of the termination requirements (manual), translation of the termination requirements in the internal formal language (automated).
Data Flow requirements specification	
<i>Objective:</i>	Define the data flow requirements for the composition problem.
<i>Automated VS Manual:</i>	Semi-automated.
<i>Input:</i>	Component service interfaces (WSDL and abstract WS-BPEL).
<i>Output:</i>	Composite service WSDL, data flow requirements (data net).
<i>Description:</i>	Specification of requirements concerning data manipulation and exchange through the data net graphical notation (manual), specification of the composite service WSDL (automated).
Composite service generation	
<i>Objective:</i>	Generate the composite service internal executable process and customer interface.
<i>Automated VS Manual:</i>	Automated.
<i>Input:</i>	Component service interfaces (WSDL and abstract WS-BPEL), composite service interface (composite service WSDL and, not mandatory, abstract WS-BPEL), control flow requirements, data flow requirements (data net).
<i>Output:</i>	Composite service executable process (WS-BPEL) and, if not given in input, composite service interface (abstract WS-BPEL).
<i>Description:</i>	Development of the composite service executable process in WS-BPEL (automated), specification of the composite service interface in abstract WS-BPEL (automated).
Composition requirements refinement	
<i>Objective:</i>	Refine the composition requirements on the basis of the automated composition results.
<i>Automated VS Manual:</i>	Semi-automated.
<i>Input:</i>	Component service interfaces (WSDL and abstract WS-BPEL), composite service interface (WSDL and abs WS-BPEL), composite service executable process (executable WS-BPEL), control flow requirements, data flow requirements (data net).
<i>Output:</i>	Composite service interface (WSDL and abs WS-BPEL), control flow requirements, data flow requirements (data net).
<i>Description:</i>	Analysis of the automated composition results and refinement of the control flow requirements, data flow requirements and composite service interface.

Fig. 2. The Main Steps of the Proposed Web Service Composition Process.

new composite service is to sell home-delivered furniture. This amounts to say that both the shipment and purchase services have been confirmed. However, since this goal cannot always be guaranteed (e.g. the shipping service may be unavailable for a given location) there is the need to specify recovery goals that must be considered in case the main goal fails. In the P&S example we do not want to book a shipment if we're not sure that the required furniture is available (and vice-versa). Thus, the recovery goal states that there must not be single commitments.

The data flow requirement specification step concerns the specification of how incoming and outgoing messages must be used by the composite service (from simple forwarding to complex data manipulation). During this step the developer also specifies messages received from and sent to the composite service user. For instance, in the P&S example we need to specify that the information on the item to be purchased that are sent to the **Producer** must be obtained from the P&S customer. While the information concerning the size of the package that the P&S sends to the **Shipper** must be the same received in the **Producer** offer.

Given the description of the component services and the composition requirements, the final step of the first phase is the development of the new composite service. The outcome of this completely automated phase is the executable WS-BPEL implementing the internal behavior of the new process and description of the interaction protocol that the new service expects its customers to follow (WSDL and abstract WS-BPEL).

During the second process phase, on the basis of the first phase outcomes, the developer can modify and gradually refine composition requirements and automatically re-generate the composite service. Requirements refinement doesn't involve only control flow and data flow requirements, but also the composite service customer interface, which can be modified and given in input to the automated composition tool as a further composition constraint.

In the rest of the paper we will describe in details each step of the proposed composition process by means of the P&S composition scenario.

3 Control flow requirements specification.

Given the description (i.e. the WSDL and abstract WS-BPEL) of the component services (e.g. the **Producer** and **Shipper** in our reference example), the first step is the formal specification of the control flow composition requirements.

The P&S service main goal is to "sell home-delivered furniture". This means we want the P&S service to reach a situation where its customer has confirmed his order, and the corresponding sub-orders have been confirmed to the **Producer** and **Shipper** services. However, it may be the case that the product is not available, that the shipping is not possible, or that the customer doesn't accept the offer. We cannot avoid these situations, therefore we cannot ask the composite service to guarantee this requirement. In case this requirement cannot be satisfied, we do not want the P&S to confirm orders without being sure that our customer accepts the offer, as well as we do not want displeased customers that have paid items that are not available or cannot be delivered. Thus, our global termination requirement must take into account the transactionality of each component service within the overall composition. The control-flow requirement should be something like: do whatever is possible to "sell home-delivered furniture" and if something goes wrong guarantee that there are "no single commitments".

In the specification of each service abstract WS-BPEL protocol we require (see Figure 3) to mark some states as successful (symbol ✓) and others as failing (symbol ×). Consider for in-

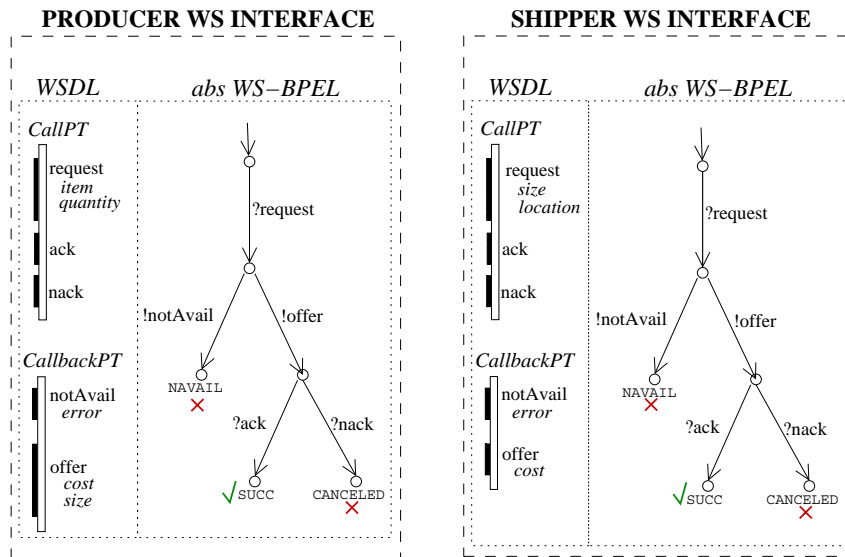


Fig. 3. PandS case study: Specification of control flow requirements

stance the Shipper service. When it receives an acknowledge message confirming the acceptance of the offer these means that a delivery contract has been defined and the protocol terminates with success. While it terminates with failure in the case of unavailability of the shipment or refusal of the delivery offer.

These annotations are used to specify the transactional requirements of the composition problem. In particular, if we consider the P&S scenario, the specification is the following:

	Producer	Shipper	PandS
Primary	✓	✓	✓
Secondary	×	×	×

The specification distinguishes two different requirements: a primary and a secondary one. The primary requirement is to reach a situation where all the services are in a successful state (in our case it models the condition “sell home-delivered furniture”). The secondary requirement (modeling the condition “no single commitments”) is to reach a situation where all the services are in a failing state. During the execution of the composite service, a state may be reached from which it is not possible to satisfy the primary requirement, e.g., since the delivery is not possible. When such a state is reached, the primary requirement fails and the secondary requirement, in our case “no single commitments”, must be guaranteed.

Notice that, while specifying control flow requirements, the developer models also the transactional nature of the new composite service. In particular, in the PandS example we are saying that the composite service is in a successful state when its interaction with the component services (Producer and Shipper) is such that both of them have reached a successful state, while the PandS is in a failing state when both the component services fail. If we consider real-world composition scenario (e.g. the Amazon-MPS example considered in [6]) we can have more com-

	<i>forwarder</i> : simply forwards data received on the input node to the output node
	<i>function</i> : upon receiving data on all input nodes, computes the function result and forwards it to the output node
	<i>fork</i> : forwards data received on the input node to all the output nodes
	<i>merge</i> : forwards data received on some input node to the output node, preserving temporal order
	<i>cloner</i> : forwards, one or more times, data received from the input node to the output node
	<i>filter</i> : receives data on the input node and either forwards it to the output node or discards it

Table 1. Elements of the data-flow requirements specification language.

plex transactional requirements. For instance it can be the case that the failure of a component service doesn't affect the success of the overall composition: the other components and the composite service can still be in a successful state.

Our approach thus provides the developer with the ability to specify with a simple tabular notation control-flow requirements that are then automatically translated into a formal internal notation that is hidden to the developer.

4 Data flow requirements specification.

The termination requirements presented in the previous section are only a partial specification of the constraints that the composition should satisfy. Indeed, we need to specify also requirements on the data flow. In order to provide consistent information, the P&S service needs to exchange data with the components and its customer in an appropriate way. For instance, when invoking the **Producer** service, the information about the requested item must be those that the P&S receives from its customer; the size information that the P&S sends to the **Shipper** must be the one received in the offer of the **Producer**. And so on.

In [8] we propose a data flow modeling language whose main aim is to allow the specification of complex requirements concerning data manipulation and exchange by means of an intuitive and easy-to-define graphical notation. In particular, data flow requirements specify explicitly how output messages (messages sent to component services) must be obtained from input messages (messages received from component services). All these requirements are collected in a diagram called *data net*, whose nodes are sources of input messages, consumers of output messages, or internal variables for temporary storage of data, and whose arcs represent flow or manipulation of data. Data nets are able to represent a variety of constraints on the flow of data, including whether an input message can be used several times or just once, how several input messages must be combined to obtain an output message, whether all messages received must be processed and sent, and so on. In Table 1 we briefly describe the basic data net elements. Refer to [8] for a complete and formal description of the language.

The approach presented in [8] requires to define data flow requirements starting from the description of the component services and of the composite service customer interface. We have extended the approach in order to deal with composition problems where the protocol of the new composite service is not specified.

Starting from the description of the messages of the component services, the developer explicitly specifies how incoming messages must be composed, manipulated or simply forwarded

to obtain outgoing messages. During this phase the developer specifies also the messages that it expects to receive from (and send to) his customer. The specification simply requires to add new input-output nodes to the data net and associate them to a newly defined operation of the composite service. When one of this node is connected, through a data net element, to other nodes, the node type is automatically derived and used to appropriately generate its corresponding WSDL message description. In this way, the WSDL description of the composite service interface, specifying the service port types, operations and messages, is automatically generated while specifying data flow requirements.

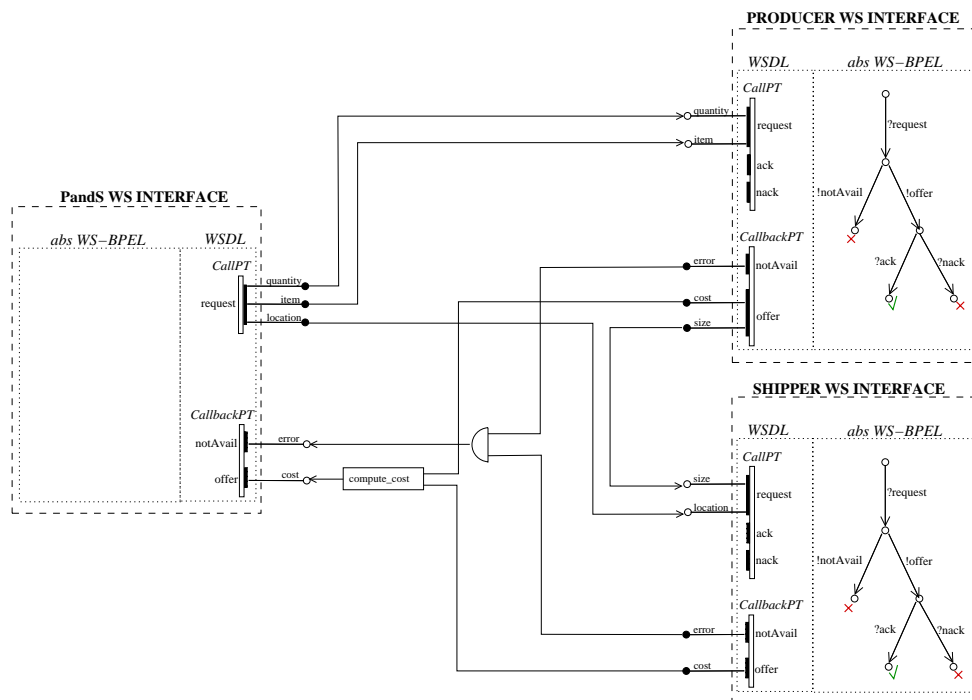


Fig. 4. PandS case study: Specification of data flow requirements

The specification of the data net for the P&S example is presented in Figure 4, which we will (partially) explain in the following example.

Example 2 (P&S data-flow requirements).

The Producer service, when invoked for a furniture purchase, expects to receive the *item* and *quantity* information in the *request* message. Intuitively, these information must be obtained by the P&S from its customer. Thus the developer specifies a new operation (*request*) whose message is composed of two parts (*item* and *quantity*). The newly defined message parts become additional input nodes of the data net and can be linked to the corresponding nodes in the Producer interface through forwarder elements. The outcome of this operation is twofold:

new data flow requirements are specified and the following definitions are automatically added to the P&S WSDL.

```

<message name = "requestMsg">
  <part name = "quantity" type = "Producer : quantityType" />
  <part name = "item" type = "Producer : itemType" />
</message>
<portType name = "CallPT">
  <operation name = "request">
    <input message = "requestMsg" />
  </operation>
</portType>

```

Similarly, to send the request message to the Shipper we must obtain the package size information and the delivery location information. The former must be the same that the P&S received in the Producer offer, and thus we define a new forwarder data net element, while the latter must be received in the customer request, and thus we add a new message part (location) to the P&S customer request and define another forwarder element. The WSDL specification of the P&S request message is automatically updated:

```

<message name = "requestMsg">
  <part name = "quantity" type = "Producer : quantityType" />
  <part name = "item" type = "Producer : itemType" />
  <part name = "size" type = "Shipper : sizeType" />
</message>

```

The information received in the Producer and Shipper offers concerning respectively the purchase cost and delivery cost, must be used to obtain the total cost of the service to be offered to the P&S customer. This requires to define a new operation (offer) to the P&S interface carrying the cost information within its message. Then a new function element is added to the data net to model the fact the the cost information sent to the customer must be obtained by applying the compute_cost internal P&S function to the cost information received in the Producer and Shipper offers. The following code is automatically added to the WSDL specification of the P&S:

```

<message name = "offerMsg">
  <part name = "cost" type = "PandS : costType" />
</message>
<portType name = "CallbackPT">
  <operation name = "offer">
    <input message = "offerMsg" />
  </operation>
</portType>

```

And so on.

Each operation arc in the datanet (e.g. compute_cost in the P&S example) refers to an internal function that the new composite service uses to manipulate data. Since our aim is to automatically generate the executable WS-BPEL code implementing the composite service, the specification of each internal function is given as an XML Path Language (XPath) expression, which is the standard language used in WS-BPEL assignments.

Example 3 (Specifying Internal Functions). Consider for instance the function compute_cost used in the datanet of Figure 4 to obtain the cost of the service to be offered to the P&S customer. The XPath specification of compute_cost, defining how to obtain the total cost by aggregating the costs received in the Producer and Shipper offers, is the following:

```

compute_cost =
  bpws : getVariableData('Producer_offer', 'cost') + bpws : getVariableData('Shipper_offer', 'cost')

```

5 Composite service generation.

The approach we apply is based on the automated composition techniques described in [7, 8]. We extended these techniques in order to support the proposed web service composition process. In particular the new automated composition approach, given the (WSDL and abstract WS-BPEL) description of the component services and the specification of the composition requirements, automatically generates both the executable WS-BPEL implementation and the abstract WS-BPEL protocol of the composite service.

The WS-BPEL processes defining the component services are modeled as state transition systems (STS). Intuitively a STS describes a dynamic system that can be in one of its possible *states* (some of which are marked as *initial states* and/or as *final states*) and can evolve to new states as a result of performing some *actions*. Actions are distinguished in *input actions*, which represent the reception of messages, *output actions*, which represent messages sent to external services, and *internal actions*, which represent internal evolutions that are not visible to external services, i.e., data computation that the system performs without interacting with external services. A *transition relation* describes how the state can evolve on the basis of inputs, outputs, or internal actions. We have defined a translation that associates a state transition system to each component service, starting from its WS-BPEL specification. We omit the formal definition of the translation, which can be found at <http://astroproject.org>. See Figure 3 for an example of the STS modeling the **Producer** and **Shipper** services.

Similarly, the data net is encoded as a STS constraining the operations that the composite service can perform to manipulate messages (refer to [8] for all the encoding details). In particular, input actions in the data net STS represent messages received by the component services, output actions represent messages sent by the component services and internal actions represent assignments that the composite process performs on its internal variables. A nice feature of our approach is that this can be done compositionally, i.e., a “small” automaton can be associated to each element of the data net, and the data net STS is obtained as the synchronized product of all these small automata.

Together, the STSs of the component services and that of the data net define the *composition domain*, i.e., they encode the constraints on the behavior of the composite service imposed by component processes and by the additional constraints on the management of data represented in the data net.

The main extension with respect to the automated composition approach presented in [7, 8] is that the specification of the composite service interface is no more a mandatory input.

If the customer interface is not specified (first phase of the proposed approach) then we encode within the composition domain the input and output operations specified in the composite service WSDL as ‘free’ input and output actions. With ‘free’ we mean that these actions can be used by the composite service whenever they are needed. To this extent we define a STS modeling the most general customer interaction protocol: for each operation specified in the composite service WSDL we define a transition that can be ‘fired’ whenever required during the interaction with the components. Figure 5 shows the STS for the **PandS** customer interface. Notice that input operations (e.g. **request** in the **PandS** example) are preceded by special internal actions (**fire_request**) modeling the fact that, unlike in the STS of the components, not only outgoing actions but also incoming actions can be activated exactly and only when needed.

If the customer interface is given as part of the composition constraints, then it is encoded as an additional STS within the composition domain (as defined in [7, 8]).

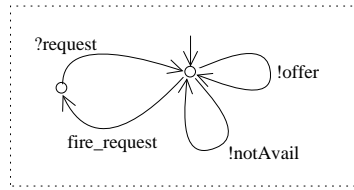


Fig. 5. *PandS* case study: STS modeling the most general customer interface

The formal specification of the composition control-flow requirements (see Section 3) is used as *control goal*. The automated generation of the composite service consists in generating a *controller*, i.e., an STS that interacts with the services encoded in the composition domain by exchanging messages with them, guaranteeing the satisfaction of the control goal.

The work in [10] shows how to generate the controller using the “Planning as Model Checking” approach, which is able to deal with large domains and complex termination requirements. This approach exploits powerful BDD-based techniques developed for Symbolic Model Checking to efficiently explore domains of large size.

Once the STS codifying the controller has been generated, it is translated into executable WS-BPEL to obtain the new process which implements the required composition. The translation is conceptually simple; intuitively, input/output actions in the controller model an interaction of the composite service with one of the component services, while “synchronizations” with actions in the STS modeling the data net correspond to manipulations of data by means of XPath expressions and assignments.

The abstract WS-BPEL modeling the composite service interface (if not already given as an input to the automated composition) is obtained by abstracting away all the interactions with the component services and the internal manipulation of data, and considering only those interactions involving the composite service customer. Automating this step is not obvious since it requires to generate a WS-BPEL code which differs from the executable one for several structural aspects. For instance, each branch of the executable composite process related to the receiving of a specific message from a component service (an `onMessage` branch of a WS-BPEL `pick` operation) must be modeled as a nondeterministic choice (a nondeterministic `case` in a WS-BPEL `switch` operation) in the user interface.

Figure 6 shows a compact representation of the automated composition outcomes for the P&S case study: the executable WS-BPEL description of the internal process and the abstract WS-BPEL description of the P&S customer interface.

The proposed customer interface is very simple: the customer sends a request message containing information about the furniture he wants to purchase and about the delivery location, then he can either receive a purchase and ship offer or a message communicating the unavailability of the requested service. Notice that the final states of the customer interface are annotated with semantic information about the outcome (successful or failing) of the overall composition. Such annotations are automatically obtained from the tabular specification of the control flow requirements (see Section 3).

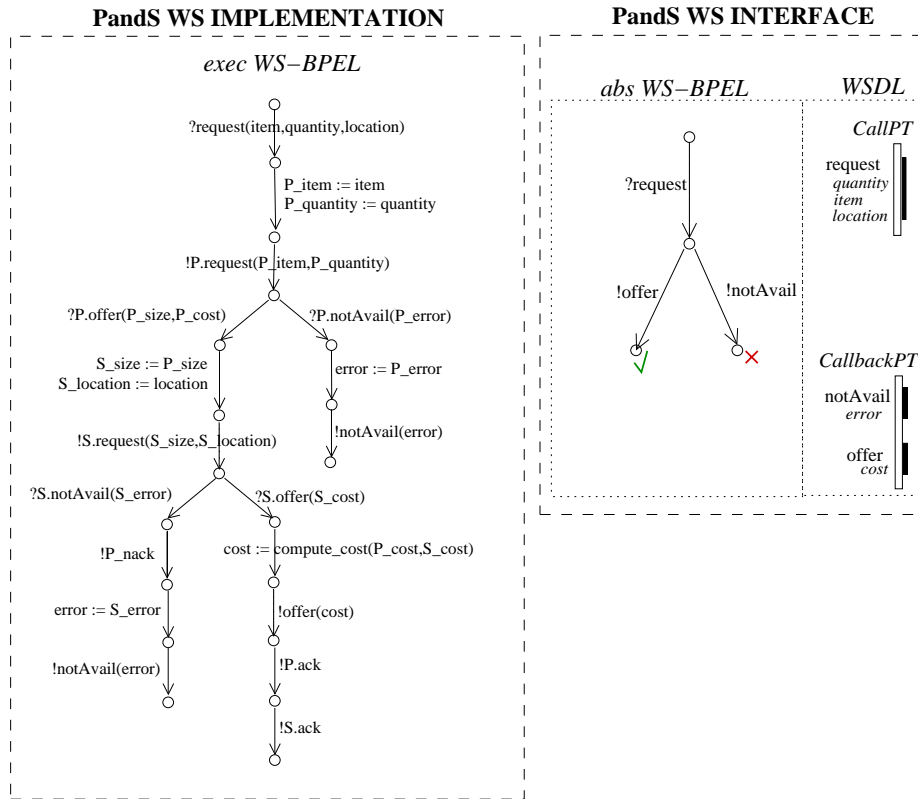


Fig. 6. *PandS* case study: Automated composition outcome.

6 Composition requirements refinement and re-composition.

Clearly the developer had in mind a more sophisticated customer interface; in particular the interaction protocol should allow the customer to evaluate the offer and then either accept it, confirming the order, or refuse it.

Instead of modifying by hand the executable WS-BPEL code of the P&S process and the abstract WS-BPEL customer interface, the developer can simply refine the composition requirements (the customer interface and control flow requirements in this case) and re-compose.

Figure 7 shows the refinement of the P&S customer interface. New operations (`ack` and `nack`) have been added to the WSDL specification and have been used within the abstract WS-BPEL to model a possible refusal of the offer. Finally the abstract WS-BPEL has been annotated with semantic information in order to model the fact that the P&S is in a successful state if it receives an acknowledgment message from its customer, while it is in a failing state if the customer refuses the offer.

The composite *PandS* process resulting from the automated re-composition with the refined composition requirements is shown in Figure 8.

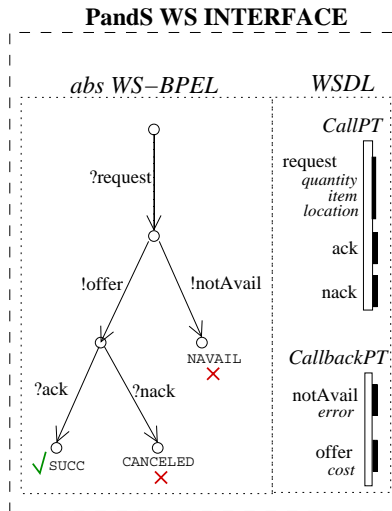


Fig. 7. *PandS* case study: Refinement of composition requirements

The refinement of the composition requirements, that for the *PandS* example involves only the user interface, can concern also the data flow and control flow requirements.

7 Conclusions and Future Works

In this paper we have presented a semi-automated composition process that, extending and improving the sophisticated composition techniques in [7, 10, 8], dramatically reduces the effort for the composition task and can scale up to realistic composition problems (e.g. the Amazon e-Bookstore described in [6]). The proposed approach automatically generates both the composite internal executable process (executable WS-BPEL) and its user interface (WSDL and abstract WS-BPEL) starting from composition requirements that can be iteratively refined on the basis of the automated composition outcomes.

The work presented in this paper has been implemented within the ASTRO toolset (<http://astroproject.org>) that has been designed as an extension of ActiveBPEL Designer [11], a commercial software for designing and developing WS-BPEL processes which is based on the Eclipse platform. The ASTRO toolset supports all the phases of web service automated composition: from the specification of control-flow and data-flow requirements (by means of graphical tools for drawing data net diagrams and specifying control-flow requirements) to the deployment and execution of the new composite service on the Active BPEL engine.

Several works address the problem of the automated synthesis of process-level compositions. However, most of them 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, 4], and of work within the semantic web community, see, e.g., [12]. For what concerns the specification of the control flow requirements that the new composite service should satisfy, most of the existing approaches specify them as reachability conditions (e.g. [13]). Some other approaches, see, e.g., [14], are

PandS WS IMPLEMENTATION

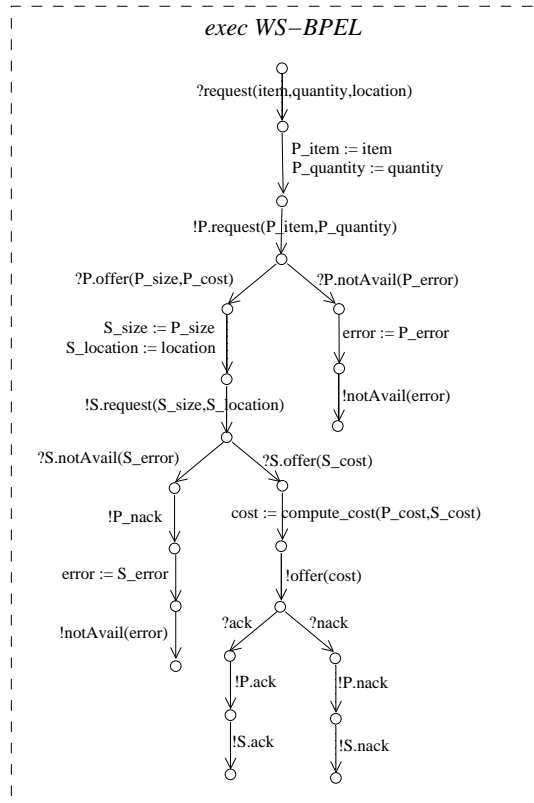


Fig. 8. *PandS* case study: Automated re-composition outcome.

limited to simple composition problems, where component services are either atomic and/or deterministic. The work described in [5], proposes an approach to service aggregation that takes into account very simple data flow requirements. As far as we know, none of the existing approaches is able to deal with real world composition scenarios and to support the entire life cycle of the proposed composition process.

Future work will include the possibility to automatically derive (part of) the data net specification starting from the semantics of the data used in the component services, and to detect failures or changes in the component services, check the realizability according to the composition requirements, and re-configure the composite process.

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., Golland, 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., Lenzerini, M., Mecella, M.: Automatic composition of E-Services that export their behaviour. In: Proc. ICSOC'03. (2003)
4. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of Services with Nondeterministic Observable Behaviour. In: Proc. ICSOC'05. (2005)
5. Brogi, A., Popescu, R.: Towards Semi-automated Workflow-Based Aggregation of Web Services. In: Proc. ICSOC'05. (2005)
6. Marconi, A., Pistore, M., Traverso, P.: Automated Web Service Composition at Work: the Amazon/MPS Case Study. In: Proc. ICWS'07. (2007)
7. Pistore, M., Traverso, P., Bertoli, P., A.Marconi: Automated Synthesis of Composite BPEL4WS Web Services. In: Proc. ICWS'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., Marconi, A., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: Proc. IJCAI'05. (2005)
10. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: Proc. ICAPS'05. (2005)
11. Designer, A.: (The Active Endpoints BPEL Designer - <http://www.active-endpoints.com>)
12. McIlraith, S., Son, S.: Adapting Golog for Composition of Semantic Web Services. In: Proc. KR'02. (2002)
13. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In: Proc. WWW'02. (2002)
14. Ponnkanti, S., Fox, A.: SWORD: A Developer Toolkit for Web Service Composition. In: Proc. WWW'02. (2002)