

# Structured Service Composition in IRS-III

Farshad Hakimpour<sup>1</sup>, Denilson Sell<sup>1,2</sup>, John Domingue<sup>1</sup> and Liliana Cabral<sup>1</sup>

<sup>1</sup>Knowledge Media Institute, The Open University, Milton Keynes, UK  
farshad@hakimpour.com, {j.b.domingue, l.s.cabral}@open.ac.uk

<sup>2</sup>Stela Group, Universidade Federal de Santa Catarina, Brazil  
denilson@stela.ufsc.br

**Abstract.** Web services are currently one of the trends in the digital economy as means to support business interoperation and collaborative processes. Semantic Web Services facilitate activities including automatic discovery and composition of Web Services. Research initiatives such as WSMO have been developing specifications for this technology. However, a model for describing all aspects of service composition is ultimately required for WSMO. This paper describes our formal model for composition of Web services in a structured approach. The proposed model complements the WSMO orchestration in IRS-III, a framework for Semantic Web Services based on WSMO specification. We also present a tool based on the above model that supports a user-guided interactive composition approach, by recommending component Web services according to the composition context.

## 1 Introduction

The stack of Web service specifications is becoming more and more comprehensive, stable and, thus, widely used. It promises a dynamic, interactive and open environment for conducting business. Consequently, business partners show increasing interest to adopt Web Services technologies. One of the challenging topics in this domain is to create Web Services, by composing existing component services, to satisfy complex problems. To address Web service composition (also called *orchestration* in WSMO [15] or *business process* in BPEL4WS [6]), we need the technology to describe a composition and to execute the composition description, while the interaction with the service client remains as simple (transparent) as a non-composite service.

Research on the Web Services composition is motivated by the need to support business interoperation and re-use –i.e. extension of available services. Challenges related to the composition process include constant changes in the business rules, high diversity and heterogeneity of Web services and the ad-hoc character of each composition.

On the other hand, semantic description of Web Services along with the necessary infrastructure facilitates automated discovery, selection, composition and execution of Web services. OWL-S [2] and WSMO [14] are two prominent initiatives that address the service composition in the domain of Semantic Web services. Both initiatives aim

at presenting specifications for semantic Web services with different perspective and background. WSMO is a European initiative based on several research projects (SDK project cluster – see <http://www.sdkcluster.org/>). It is developing specifications for service descriptions in order to support composition as well as discovery, selection, and execution of Web services. OWL-S is an initiative in the framework of DAML program. OWL-S initiative started long before WSMO and provides a service composition model [2]. WSMO specification for composition [15] is yet to further develop in details. Therefore, a model for describing all aspects of service composition is required for such complete specification.

This paper describes a model for Web service composition developed in the context of the IRS-III (Internet Reasoning System [3]). IRS-III is a framework and implemented infrastructure that supports the creation of Semantic Web Services according to WSMO. The model presented in this paper is based on structured design paradigm and extends WSMO ontology to propose a composition component. The necessary modules are implemented and running as part of IRS-III. In addition, we present a tool that supports a user-guided, interactive composition process, whereby Goals are recommended in each step of a composition.

The rest of the paper is organized as follows. In section 2, we present a short background for the paper. Section 3 shows the formal definition for the structured approach to Web service composition. Section 4 presents an example scenario. Section 5 describes our composition model. In Section 6, we briefly point out the features of the orchestration engine. Section 7 presents our tool for semi-automatic composition of Web services. Finally, in section 8, we present the conclusion and the future work.

## **2 Background**

In the following, we present a brief introduction to WSMO specification for Semantic Web Services and IRS-III framework.

### **2.1. WSMO**

The Web Service Modeling Ontology (WSMO) [14] is a formal ontology for describing the various aspects related to Semantic Web Services. The main components of WSMO are Goals, Web services, Ontologies and Mediators.

Goals represent the types of objectives that users would like to achieve, describing the state of the desired information space and the desired state of the world after the execution of a given Web service. WSMO Web services specifications describe the functional behavior of an actual Web service. WSMO describes Web services by their capabilities and interfaces. A Web service may be selected by discovery process and then executed when a Goal is required. The discovery can be done by matching the capabilities (i.e. precondition, effect, etc. see [14]) of a Web service with those of required (in a Goal). The interface description contains two closely related notions of choreography and orchestration. Choreography describes information required to

interact with a Web service. Orchestration, which is the focus of this work, can contain information describing a composite Web service. Unlike choreography, orchestration provides the necessary details for the execution of all the component services and may be a proprietary item for the provider –i.e. not accessible to others.

Ontologies provide the basic glue for semantic interoperability and are used by the three other components. Mediators specify interoperability mechanisms and provide the means to link the three components described above. The incorporation of mediators in WSMO facilitates the clean separation of different interoperability mechanisms.

## **2.2. IRS-III**

IRS-III [3] is a framework and implemented infrastructure for Semantic Web Services that implements the WSMO descriptions. IRS-III has four main classes of features that distinguish it from other work on Semantic Web Services. Firstly, it automatically transforms programming code into a Web service, by creating an appropriate wrapper. Hence, IRS-III makes it easy to make existing standalone software available on the net, as Web services.

Secondly, users of IRS-III directly invoke Web services via Goals, supporting capability-driven service discovery and invocation. In IRS-III, Goals are logical description of problems and Web services are logical description of solutions. An invocation of a Goal results in a discovery process, matching solutions available for a problem (see [3] or [14] for details).

Thirdly, IRS-III is programmable. IRS-III users can substitute their own semantic Web services for some of the main IRS-III components. Finally, IRS-III services are Web service compatible –standard Web services can be published through the IRS-III and any IRS-III service automatically appears as a standard Web service to other Web service infrastructures.

## **3 Formal Definitions**

This section provides formalization with clear semantics to represent compositions. A suitable model for service composition should contain three major sets of information:

- Information about component services;
- Information about the order of execution of the component services;
- Information about the data binding between services or data flow.

The components taking part in a composition may be statically bound to the composition as a Web service or described as a Goal –i.e. dynamically assigned by a discovery process during the execution time. IRS-III allows both methods in the description of a composition. Our formalism should present features required for defining the execution order or control flow of composition (i.e. algorithm). It should

also provide mechanism for defining data flow in the composition. In the following, we provide a formal specification of our composition model.

**Definition:** A structured composition  $O$  is a two tuple  $\langle T, \Delta_T \rangle$  where:

- $T$  is a rooted ordered tree [13] called *composition tree* with nodes:  $X_T$ , edges:  $E_T$  and the root:  $\chi_0 \in X_T$ .  $S_T \subseteq X_T$  is the set of all leaves in  $T$  and  $C_T \subset X_T$  is the set of all internal vertices (note that,  $X_T = S_T \cup C_T$ , also  $S_T \cap C_T = \emptyset$ ). Members of  $X_T$ ,  $S_T$  and  $C_T$  are called *composition components*, *service components* and *control components*, respectively. For every control component  $c \in C_T$ , one and only one of the following is true: *Seq(c)*, *While(c)*, *Con(c)* or *If(c)*. The children of a composition component  $c \in C_T$ , denoted by *Children(c)*, set out an ordered list. (See Figure 1 for an example of a composition tree, where control components are shown by ovals and service components are illustrated by boxes).
- $\Delta_T$  is a directed graph [13] called *data flow* defined on  $T$ , with vertices  $i_\Delta$ ,  $o_\Delta$ , and members of  $X_T$  (composition components of  $T$ ); and the directed edges  $B_\Delta$ , where  $B_\Delta \subseteq \{ \langle s_i, t_j \rangle : (s_i \in S_T \vee s_i = i_\Delta), (t_j \in X_T \vee t_j = o_\Delta) \}$ . A member of  $B_\Delta$  is a binary tuples called a *binding* between  $s_i$  and  $t_j$ , where  $s_i$  is the source of the binding and  $t_j$  is the target of the binding. (See Figure 2 for an example of a data flow).

### 3.1. Semantics

Service components ( $S_T$ ) in a composition represent either a Goal or a Web service (an *invocable*). At the implementation level, service components are wrappers that hold information about the bindings around Web services or Goals, while for a composition designer they represent Web services or Goals.

Control components ( $C_T$ ) provide the ability to define the order of execution (i.e. algorithm). Control components are of four different types: sequence, concurrent, if-then-else and while. The functional semantics of a control component (i.e. the behavior of orchestration engine associated with each control component) depends on their type. So far, we implemented four above mentioned control components. Semantics of the above control components are considered intuitive, here ([10] present the semantics of the control components for OWL-S by Petri Net, that includes the above four components). The only issue that may require clarification here is the synchronization in case of concurrent component. The execution of a concurrent element is finished when execution of all its child elements are finished.

A major set of required patterns for building compositions can be treated by these components. However, they require extension to cover more patterns (as we discuss in section 8).

$B_\Delta$  represents a set of bindings describing data flow between components, or between components and input or output (arrows in Fig. 3 illustrate data bindings). The orchestration engine will run a mediator when encountered with each binding. The engine activates mediators bound to every service the moment the service provides the output. Then, the orchestration engine, either buffers the result of the mediation for further use by other services, or passes it as output of the composition.

In sections 5 to 7, we discuss how bindings can use mediators to transfer data between services.

### 3.2. Normalization and Deadlock Validation

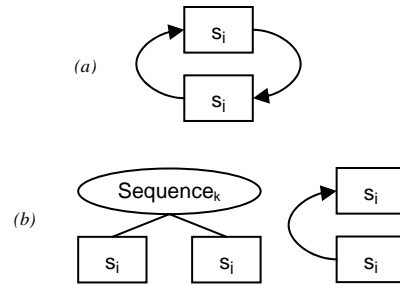
To avoid compositions containing combinations such as sequence of other sequences, we define the notion of normalized composition tree.

**Definition:** A composition tree  $T$  in a structured composition  $O\langle T, \Delta_T \rangle$  is normalized, if the following hold:

$$\begin{aligned} \forall c \in C_T: (Seq(c) \Rightarrow (\forall \chi \in Children(c): \neg Seq(\chi))) \\ \forall c \in C_T: (While(c) \Rightarrow (\forall \chi \in Children(c): \neg Seq(\chi))) \\ \forall c \in C_T: (Con(c) \Rightarrow (\forall \chi \in Children(c): \neg Con(\chi))) \end{aligned}$$

We simplify such cases by a normalization process. To normalize a composition tree, the child component  $\chi$ , must be removed from the tree and all its children should be added as children of  $c$  in the tree at the same position in the children list, while they keep the same order. Note that normalization only optimizes the composition tree and the execution of a composition and does not have any effect on the result of a composition.

Deadlocks can occur during the execution of a composition. For example, a deadlock for two component services may happen when each one is waiting for the output of the other one (Figure 1a), or when one is waiting for the output of the other when it is to execute in a later stage in a sequence (Figure 1b).



**Fig. 1.** Illustration of example causes of deadlock-prone compositions.

In the following, we define the notions of deadlock-safe and deadlock-prone for a composition. First, we define the dependency graph where it represents the dependency of every service component to others. Such dependency is justified either because of reliance on the data provided by another service, or because of the order of execution.

**Definition:** Dependency graph of a composition  $O\langle T, \Delta_T \rangle$  is a directed graph  $D_O$ , where nodes are all the service components of  $O$ . The edges of the dependency graph are made by walking through the composition tree  $T$ , in breadth first manner [13], and modify it as following:

- If the current node  $c$  is sequence or while {if  $seq(c) \vee while(c)$ }:
  - 1) a directed edge is inserted from every node to its next node in the list of its children,  $Children(c)$ ;
  - 2) if  $c$  already depend on a component  $\chi$  ( $\exists e\langle \chi, c \rangle$ ), then a new directed edge representing the dependency between  $\chi$  and the first component in  $Children(c)$  is added ( $insert(e'\langle \chi, first(Children(c)) \rangle)$ ) and  $e$  is removed;

- 3) if a component  $\chi$  already depends on  $c$  ( $\exists e \langle c, \chi \rangle$ ), then a new directed edge representing the dependency between the last component in  $Children(c)$  and  $\chi$  is added ( $insert(e' \langle last(Children(c)), \chi \rangle)$ ) and  $e$  is removed;
  - 4) Finally,  $c$  is removed.
- If the current node  $c$  is concurrent or if-then-else  $\{Con(c) \vee if(c)\}$ :
    - 1) if  $c$  depend on a component  $\chi$  ( $\exists e \langle \chi, c \rangle$ ), then for every component child of  $c$  a new directed edge representing the dependency between  $\chi$  and the components in  $Children(c)$  must be added ( $\forall \chi_i' \in Children(c), insert(e_i' \langle \chi, \chi_i' \rangle)$ ) and  $e$  must be removed;
    - 2) if a component  $\chi$  depend on  $c$  ( $\exists e \langle c, \chi \rangle$ ), then for every child component of  $c$  a new directed edge representing the dependency between child component and  $\chi$  must be added ( $\forall \chi_i' \in Children(c), insert(e_i' \langle \chi_i', \chi \rangle)$ ) and  $e$  must be removed.
    - 3) Finally,  $c$  and the edges to its children are removed.

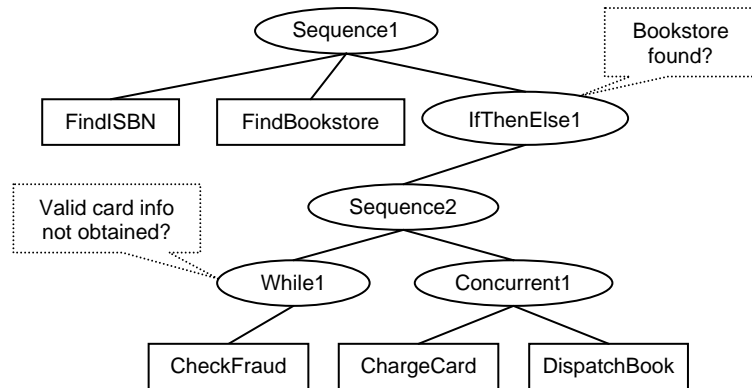
After the above modifications, the edges in data flow graph,  $\Delta_T$ , where both source and target are service components are added to the resulting graph,  $D_O$ . An example of a dependency graph for the example in section 4 is illustrated in Figure 3.

**Definition:** A composition  $O \langle T, \Delta_T \rangle$  is *deadlock-safe* if its dependency graph contains no directed cycle.

A deadlock-safe composition is guaranteed that will not enter a deadlock state. The composition in section 4 is a deadlock-safe as the graph in figure 3 does not contain a directed cycle.

**Definition:** A composition  $O \langle T, \Delta_T \rangle$  is *deadlock-prone* if its dependency graph contains at least one directed cycle.

If all service components in a directed cycle are stateless, then that cycle is bound to cause a deadlock. However, if one or more of the services in the directed cycle is stateful [4], then more consideration of the states of the services is required to determine if they cause a deadlock.



**Fig. 2.** An example of a composition tree for buying a book in a virtual bookstore.



FindBookstore, where the dependency is result of both the composition tree and the data flow graph.

## 5 Composition Modeling

In this section, we briefly describe how we model the above formalization in a knowledge representation language (in OCML [9]) and how one can compose Web services in IRS-III by this model. We particularly emphasize on the role of mediators to describe data flow. The structured composition in IRS-III is defined as an extension of WSMO *Orchestration* –i.e. it is a subclass of the WSMO *Orchestration* class [14].

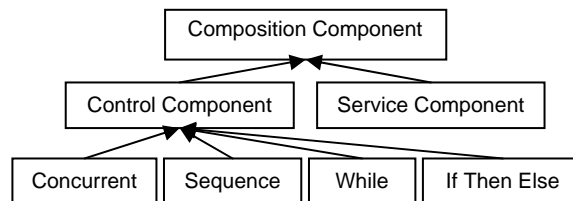


Fig. 5. Hierarchy of composition component types in the model.

The composition model is made of two types of components, namely control components and service components. Figure 5 shows the hierarchy of the composition component types. The control components provide the capability to define the control flow of the composition and they are of different types: sequence, concurrent, while and if-then-else. Figure 2 illustrates control components by ovals and service components by boxes. Listing 1 shows the definition of the root component for the example in Figure 1. (All listings in this section are simplified for illustrative purposes. Additionally, we only show examples of instance definition and avoid presenting class descriptions due to lack of space.)

**Listing 1.** An example of a structured orchestration and its root component from Figure 1.

```

book-selling-orchestration
  (structured-orchestration)
  has-root-component :value sequence1
sequence1 (sequence)
  has-components
    :value (find-isbn-SC
            find-bookstore-SC
            if-bookstore-found)
  
```

A service component is actually a wrapper that keeps the necessary information about the data bindings for an invocable Goal or Web service. The example in Listing 3 shows how a service component is defined. Service components wrap exactly one invocable description. The invocable may be located locally at the same server where the composition is stored or on any other IRS-III server (assuming that the invoker's security credentials are accepted). That allows us to compose services located on several IRS servers.

All data bindings are defined by means of mediators. That is, not only we can map data between component services, but also, perform other processes such as conversions and calculations on data between them, by means of mediators for that purposes. In the service component in Listing

**Listing 2.** Description of a service component in the “sequence” described in Listing 1.

```
find-bookstore-SC (service-component)
  has-invocable-description
    :value find-isbn-goal
  has-internal-bindings
    :value (binding-to-dispatch-book
            binding-to-charge-card
            binding-to-if-bookstore-found)
  has-binding-to-composition-output
    :value output-binding2
```

2, we can see three bindings to other service components and one binding to the output of the composition. In general, three sets of mediators can be bound to every service component: mediators between the output of a service and other services in the composition (internal bindings); mediators between the service and the output of the composition (output bindings); mediators between the input to the composition and the current service (input bindings).

Listing 3 shows one of the internal bindings and its mediator description for the service described in Listing 2. The internal binding specifies that the mediator `mapping-mediation01` should be applied to the output of the service component and produce the required input for `service-component-find-bookstore`.

Use of mediators is an important feature of IRS-III. IRS-III supports several types of mediators that are outside the scope of this paper. The only type of mediators that is used in compositions and executed by the orchestration engine is *Goal Invocation Mediator* (GInv). Goal Invocation Mediator inherits the following roles from super class Mediator described in IRS-III (based on WSMO description [14]):

- `has-source`: specifies the source Goal types that their outputs can be mediated by the mediator.
- `has-target`: specifies the target Goal type that its inputs can be provided after mediation by this mediator.
- `has-mediation-service`: a Goal or a Web service that performs the mediation. All mediators in IRS-III are also Web services that are invoked through the IRS server.

**Listing 3.** An example of a binding and its mediator for the service component in Listing 2.

```
binding-2-charge-card (internal-binding)
  uses-mediator
    :value mapping-mediation01
  to-component-service
    :value charge-card-SC
mapping-mediation01 (GInvMediator)
  has-source :value find-bookstore-goal
  has-target :value charge-card-goal
  has-mediation-service
    :value mapping-mediation-goal
```

The mediator in Listing 3 shows that the output parameters taken from `FindBookstore` (i.e. `book-price`) are passed to the mediation service and then provided as input for `ChargeCard` (i.e. `cost`). Just like service components being wrappers around Goals and Web Services that keep required information for a composition, bindings are wrappers around mediators that are used to keep necessary information for a composition. Bindings provide additional parameters for the adoption of a generic *GInv* mediator for a specific use in a composition. For example, if a constant

input value may be provided to a generic mediation service in a specific composition. (we skip further description of bindings due to lack of space).

If-then-else and while components are defined by a condition and a set of components. For an if-then-else the maximum number of components is limited to two and the first component (then) is obligatory while the second (else) is optional. To define a conditional control component one should make sure that the necessary parameters for the condition are bound (mediated) to the conditional component. Just as in case of the while and if-then-else in Figure 2.

We enhanced our data flow model by providing a specific type of binding that controls the data flow, called *conditional binding*. Using this feature different bindings (and mediators) may be activated according to the result of the evaluation of a condition. Conditional bindings are often required when using concurrent control components.

## 6 Building and Executing Compositions

The structured composition capabilities of IRS-III are provided in two modules. The first module presents a Java API to build Web service compositions. The API offers the necessary features to build a composite service and store it on IRS-III server, as well as, retrieving a composition from the server and editing it. It also provides capabilities to normalize and validate the composition. The validation includes finding directed cycles in the dependency graph (Section 3.2) and checking if necessary bindings for all inputs to component services are provided. One can define different types of mediators and add them to the mediator types as a subclass of the Goal Invocation (GInv) Mediator. For every mediator one can define an API to facilitate the ease of building mediators in Java. Listing 4 shows the Java statements to describe the classes in Listing 2 and 3.

**Listing 4.** The following Java code results in the OCML descriptions in Listings 2 and 3, which can be used by the IRS-III server.

```
ServiceComponent findBookstore = new
    ServiceComponent(irsServerDescription,
        ontologyName,
        findBookstoreInvocable);
findBookstore.setInternalBindings(
    findBookstore2ChargeCard,
    ChargeCard);
MappingMediator findBookstore2ChargeCard = new
    MappingMediator(findBookStore,
        ChargeCard,
        mapping-mediation-service,
        ontologyName,
        serverDesc);
findIsbn2FindStore.setBinding("has-price",
    "cost");
```

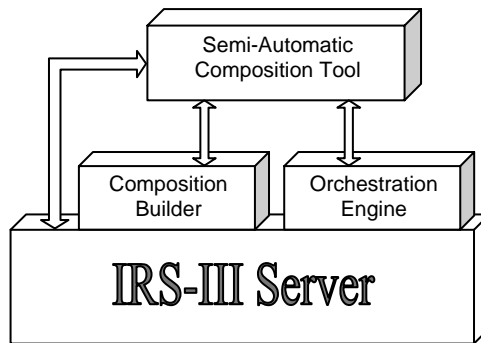
Second module provides the orchestration capabilities. This module provides an API to execute a service composition and control its execution. When a composite service is invoked, the orchestration engine executes the composition description of the service. The engine invokes a Goal for every service component in the orchestration according to the specified order and only after the necessary data for the service is provided. Orchestration engine executes a set of mediators after the output

of a component service is ready and when an input to the composition is provided. These mediators prepare consumable set of data for component services or for the output of the composition. If a service component requires data that is not yet available the thread running the service halts and notifies the invoker to provide the required data.

The structured orchestration engine is stateful [4]. As a result, invocation of a composite service not only supports the stateless interaction, but also, interaction through a set of messages at different states during the execution. The pattern of interaction for a stateless service is passing a set of input when invoking the service and receiving an output at the end of execution. The interaction with a stateful service requires exchanging messages between the invoker and the orchestration engine during the execution.

The orchestration engine provides the capability to interact with the services during the execution at different states. That is, the invoker can send input and receive outputs during execution. Finally, we also provide necessary features for monitoring the status of every service component as well as monitoring and controlling the execution of the composition itself.

The Composition tool described in Section 7 is built on top of the both modules described above. The motivation to separate the design and orchestration module is the fact that those tools that only invoke and execute a composition by IRS-III do not require to have access to the composition builder module.



**Fig. 6.** Building a compositions and their execution is provided by two separate modules.

## 7 Semi-Automatic Composition Tool

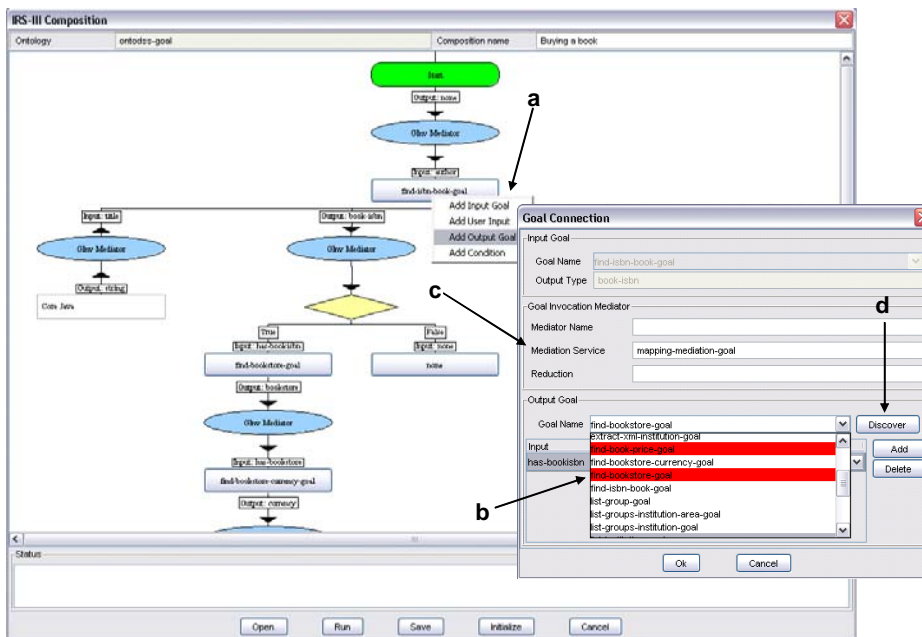
In this section, we introduce a graphical tool developed as part of IRS-III and based on our composition model. This tool supports users in designing dynamic compositions in IRS-III by recommending Goals according to the context at each step of designing the composition.

The full automation of the composition process is still the objective of ongoing research activities. As a first step to achieve this objective, we have developed a semi-automatic composition tool by supporting the user during the process of designing the composition [12]. Non-functional expectations on a service composition are expressions of human will and consequently should be given by the users instead of letting composition engines to guess or randomly assign the right service [8]. We chose an approach similar to those described in [8], and [12] in the sense that a human holds control of the definition of the composition, but the laborious work such as the discovery and invocation of services according to an abstract representation of

users' requirements is carried out by the machine. However, our approach introduces additional features such as the dynamic invocation of Web services, control operators and mediation.

Figure 7 depicts the composition tool and some of its functionalities. The tool guides users in a step-by-step composition process by selecting Goals, mediators and control flow operators. The composition starts with the selection of the first Goal, when the user receives a list containing all the Goals defined in the IRS-III Server. The user can select a Goal scrolling the list or use the discovery functionality to search for Goals by defining some search criteria, using a logical operator and identifying properties and correspondent values. The search criteria are translated into an OCML expression and processed within the IRS-III Server.

Using the composition tool, users can add Goals. Each Goal can receive input from or provide output to other existing component Goals. Goals can receive input from more than one source. For instance, a Goal that have three inputs can have one input entered from the main input to the composition and the remaining inputs from other component Goals. Users can define the values for the inputs of the selected Goals in either design or orchestration time. Finally, users can add if-then-else control operators to the composition. The interactive process is supported by the tool, which in each step recommends Goals by matching the inputs and outputs of the Goals that were previously selected considering also the subsumption of the input and output types.



**Fig. 7.** Illustration of the partial definition of the buying book scenario in the composition tool. Users interactively define a composition (a) receiving recommendations according to the automatic match of inputs and outputs of Goals (b). Users also can define mediators (c) or call the discovery functionality (d).

One important characteristic of our tool is that it enables users to select mediators to handle heterogeneities between Goals. Mediators can solve mismatches between different parties in the data, protocol and process levels. The GInv mediator presented in the last sections is an extension of WSMO mediator specifically added to IRS-III to support flexible mappings in our composition model.

We consider mediators a basic requirement to support business interoperations. The adoption of mediators gives more flexibility to users, since it is inevitable to select services defined and implemented by different parties while building a composition.

Once a composite service is defined, the composition tool instantiates the workflow using orchestration engine. During the orchestration, the user should enter required values for inputs to Goals. Inputs are required if their values are not specified in design time or they are not provided by other Goals. In addition, users can monitor the status of the orchestration by examining the status bar provided in the composition tool. The orchestration reports to the users the transformations performed, conditions satisfied and values that should be entered to complete the orchestration. Finally, the result of the composition is presented to the user in a separated dialog.

## **8 Discussion and Related Work**

Our structured composition model follows a structured system analysis and design approach. In this approach, the composition designer breaks a Goal to sub-goals that matches to one or more Web Services. It is a process-oriented approach as in contrast to the object-oriented approach. Obviously, Web services are conceptually closer to processes rather than objects. The structured approach is used in other service composition models such as in BPEL4WS [6] and OWL-S [2], and is also used in workflow modeling [7].

An advantage of our approach is the use of Goals, while other initiatives compose only Web services (such as BPEL4WS [6]). This feature provides a certain level of dynamism for compositions because suitable Web services are discovered at execution time. IRS-III provides the possibility of discovering a Web service for a Goal at the execution time. For instance, in the example presented before, one may use a Goal for a service such as FindISBN where the relevant service can be found on the fly, depending on the preconditions on book's publisher. However, one can also use a specific Web service, for example, when one obviously prefers using a particular proprietary Web service for charging a customer's credit card (ChargCard).

Another advantage of our tool is the use of mediators. All data bindings in our tool are using mediators. In addition, it makes our visualization of the transformations in a composition more comprehensible, as compared to the approach used by OWL-S [2] and BPEL4WS [6], where no distinction between mediators and Web services are made.

In our composition model, we modeled the control flow and the data flow separately with minimal inter-dependency. This allows us to combine different models for control flow with different models for data flow. We have experienced

using different models of control flow (structured and state-based) with the same data flow model [11].

Our composition model and orchestration engine support most workflow patterns introduced in [1]. Many patterns are supported by our control components. Basic patterns such as Sequence or Parallel Split are directly supported, and some branching or loop patterns such as Exclusive Choice, Multi-choice and Cycles are supported indirectly, by a combination of existing control components. Supporting more advanced patterns require further development of the orchestration engine. For example, supporting patterns where unknown number of instances of a service is initiated at run time (such as patterns 14 and 15) require further investigation as these patterns can cause undetermined behavior of the current orchestration engine.

The basis of the structured model presented in [5] is similar to our model, however, the model in [5] is not formalized. Furthermore, the composition model and the tool in [5] do not take into account the semantic aspects of Web services, and therefore the on the fly discovery using Goals, as in our approach.

Our composition tool is a step towards an automatic composition tool. The composition tool suggests Goals according to a composition context in an interactive process. CAT [8] uses a similar approach and integrates planning techniques to track relations among the composition components. However, CAT does not support the use of mediators as well as control components (similar to [12]). The OWL-S composer [12] supports users in the composition by narrowing the list of Web services based on the match of their inputs and outputs. We adopted a more flexible process, by recommending Goals through the match of their properties but allowing users to define mediators between Goals that do not match.

## **9 Conclusion and Future Work**

Web service composition is a new but essential issue to enhance B2B e-commerce over the Internet. This paper discusses a model and correspondent formalism for Web service composition in IRS-III. We present an approach to describe a composition by extending the existing WSMO ontology associated with orchestration [15] (currently under development) and describe different aspects of it. Furthermore, we created a composition tool based on the model. This tool facilitates the construction of compositions and it is a step towards an automatic composition tool. While the full automation of the composition is still subject of ongoing research, we provide the necessary means for users to define their expectations on a service composition through our composition tool. The composition tool suggests Goals according to a composition context and supports definition of mediators and control operators in an inter-active composition process. Our composition model and tool are compliant with WSMO specifications.

As for future work, we are considering enhancing our composition model by adding features to support further workflow patterns described in [1] (see Section 8) as well as error handling and compensation. We are also investigating semantic aspects related to the automatic composition following an approach based on

parametric design and incorporating planning techniques to our composition tool to support the partial automation of the composition process.

## Acknowledgement

This work is supported by the Data, Information and Process Integration (DIP) project funded under the European Union's IST program (FP6-507483); the Advanced Knowledge Technologies (AKT) project funded by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01; and the CNPq, Brazil in form of a scholarship held by Mr. Sell.

## References

- [1] W.M.P. van der Aalst, et al., "Workflow Patterns", *Distributed and Parallel Databases*, 14(1):5-51, 2003.
- [2] DAML Services Coalition, "Web Service Description for the Semantic Web", *Proc. of the First Int'l. Semantic Web Conf. (ISWC)*, [www.daml.org/services/owl-s/ISWC2002-DAMLS.pdf](http://www.daml.org/services/owl-s/ISWC2002-DAMLS.pdf), 2002.
- [3] J. Domingue, et al., "IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services", *Proc. of the Workshop on WSMO Implementations*, 2004.
- [4] I. Foster, et al., "Modeling Stateful Resources with Web Services", 2004.
- [5] D. Ganesarajah and E. Lupu, "Workflow-based composition of web-services: a business model or a programming paradigm?", in *Proc. of the 6th Int'l Enterprise Distributed Object Computing Conf. IEEE Computer Society*, ISBN 0-7695-1742-0, 2002.
- [6] IBM Corporation, *Business Process Execution Language for Web Services*, [www-128.ibm.com/developerwo-rks/library/ws-bpel/](http://www-128.ibm.com/developerwo-rks/library/ws-bpel/), 2003.
- [7] B. Kiepuszewski, A.H.M. ter Hofstede and C Bussler, "On Structured Workflow Modelling", *Proc. of the 12<sup>th</sup> Int'l. Conf. on Advanced Information Engineering*, Springer Verlag LNCS 1789, pp. 431-445, 2000.
- [8] J. Kim, M. Spraragen and Y. Gil, "An Intelligent Assistant for Interactive Workflow Composition", *Proc. of the Int'l Conf. on Intelligent User Interfaces*, Portugal, 2004.
- [9] E. Motta, "An Overview of the OCML Modelling Language", *the 8th Workshop on Knowledge Engineering Methods and Languages*, 1998.
- [10] S. Narayanan and S. A. McIlraith, "Simulation, Verification and Automated Composition of Web Services", *Proc. of the Int'l World Wide Web conf.*, pp. 77-88 2002.
- [11] D. Sell, et al., "Interactive Composition of WSMO-based Semantic Web Services in IRS-III", *AKT Workshop on Semantic Web Services AKT-SWS04*, 2004.
- [12] E. Sirin, B. Parsia and J. Hendler, "Filtering and selecting semantic Web services with interactive composition techniques". *IEEE Intelligent Systems*, 19(4), 2004, pp. 42-49.
- [13] Wikipedia, the free encyclopedia, *Graph Theory*, [http://en.wikipedia.org/wiki/Category:Graph\\_theory](http://en.wikipedia.org/wiki/Category:Graph_theory)
- [14] WSMO, "Web Service Modeling Ontology-Standard", <http://www.wsmo.org/2004/d2/>, 2004.
- [15] WSMO, "Ontology-based Choreography and Orchestration of WSMO Services", <http://www.wsmo.org/2004/d14/>, 2004.