A Framework for the Semantic Composition of Web Services Handling User Constraints

Youssef Gamha, Nacéra Bennacer Seghouani, Guy Vidal-Naquet, Béchir Ayeb, Lofti Benromdhane

A Framework for the Semantic Composition of Web Services Handling User Constraints

Y. Gamha[2], N. Bennacer[1], G. Vidal-Naquet[1], B. Ayeb[2], L. B. Romdhane[2]
[1]*SUPELEC, Joliot-Curie road, Gif-sur-Yvette, 91192, France*
[2]*PRINCE Research Unit, Faculty of Sciences Monastir, Tunisia*
Youssef_gamha@yahoo.fr, nacera.bennacer@supelec.fr, Guy.Vidal-Naquet@supelec.fr, ayeb_b@yahoo.com, lotfi.ben.romdhane@usherbrooke.ca

Abstract

In this work, we present a framework for the semantic composition of web services based on Statecharts and uniform community service descriptions. Our model is a two step process. In the first step, we derive the execution model of the user's query. The execution model is specified in Statecharts formalism; whereas the user's query is described in OWL-S. Therefore, a mapping from Statecharts formalism to OWL-S is developed. In the second step, we instantiate the developed execution model through invocation of available e-services instances. Hence, and a result, we obtain an execution plan (said also strategy) satisfying user constraints. The key features of the proposed framework could be summarized as follows. First, unlike other existing languages, using OWL-S enables the semantic description of e-services. These semantics are taken into consideration in our composition strategy. Second, the user constraints (or preferences) are taken into account during composition and are expressed as a finite set of logical formulas with the Knowledge Interchange Format (KIF) language.

1. Introduction

Nowadays, e-services are becoming more and more of common use in everyday life. Unfortunately, they are still unable to satisfy all users' requests which, often, are too complex to be satisfied by existing single (or atomic) e-services. Hence, we have no escape but compose a set of existing services to satisfy such requests. Intuitively, Web service composition could be defined *as the process that specifies how to select, combine and execute a set of available web services to obtain a composite service which fulfils the user's request*. A central problem, addressed by automated web services composition, is the inherent structural and behavioral heterogeneity of current web services caused by their strict autonomy.

Several efforts, at both academic and industrial levels, were devoted to the automatic composition of web services in order to realize such a new generation of services. These efforts aim to build current web service technology around SOAP, WSDL, and UDDI which provide standardized foundations respectively for service discovery, description, and messaging protocols. Nevertheless, these languages are insufficient to achieve a real automation of web services composition.

An emerging industry initiative to standardize some aspects of web services composition is BPEL4WS [19]. BPEL4WS is based on a human driven process, and focuses on specifying manually the coordination between multiple web services described in WSDL language and where the bindings between them are known a priori. Moreover, BPEL4WS provides different constructs for processes and data flow, but it does not support the notion of world state. On one hand, the automated composition of web services requires representing their abilities in a way that is unambiguous and interpretable by a machine. On the other hand, it involves a reasoning system which selects, combines and executes compatible web services by resolving constraints and by matching inputs, outputs, preconditions and effects to achieve the overall goal of the composition. The semantic descriptions of web services make them machine interpretable and offers agents the possibility to automatically compose different services for a new composite service. The ontology Web Language for services OWL-S [20] provides richer semantic specifications with different views of the capabilities of web services. In particular, it provides a functional view of the service as a process which describes both the information transformation which results in the production of outputs from a set of inputs, and the state transformation that results in the generation of the effects starting from a state that satisfy the preconditions.

This paper presents a framework for the semantic and automatic composition of e-services. This framework is based on a distributed architecture; on semantic uniform based-ontology, and community service descriptions. The concept of community has several appealing properties. First, it provides a mean to define services not only with

the same language but also with an ontology which is specific to the community. Second, the community is an integrator of services. This means that it offers a unified, homogeneous and a consistent access interface to existing heterogeneous e-services exhibiting similar functionalities. In addition, the use of the OWL-S language [20] allows a rich expressivity for the semantic description of services and user constraints. In the proposed framework, the composition is realized in two steps. In the first step, we derive the execution model of the user query. The execution model is specified in Statecharts formalism; whereas the user's query is described in OWL-S. Therefore, a mapping from Statecharts formalism to OWL-S is defined. The use of OWL-S and Statecharts is motivated by the fact that OWL-S allows a semantic description of web services and user constraints [20]; whereas, Statecharts are well-established tools offering *all* required control-flow constructs. In the second step, we instantiate developed execution model through invocation of the available e-services instances. Hence, and a result, we obtain an execution plan (also said strategy) satisfying user constraints. The key features of the proposed framework could be summarized as follows. First, unlike other existing languages, the use of OWL-S enables the semantic description of e-services. These semantics are taken into consideration in our composition strategy. Second, the user constraints (or preferences) are taken into account during composition and are expressed as a finite set of logical formulas with the Knowledge Interchange Format (KIF) language. Consequently, during the composition, not all e-services instances are considered, but only those satisfying the specified constraints. This has several interesting properties as reducing the search space and delivering a "quality"-service since "user preferences" are not neglected. In addition, when several service instances meet the needs of the user's request, then it is up to the user to select an instance for the composition strategy[1]. Finally, composition and coordination of the execution of a composite service is distributed across several components called *composer agents* (*or community composer agents*).

The rest of this paper is organized as follows. In section 2, we outline related research. Section 3 presents our composition framework. In section 4, we present user constraints descriptions; and in section 5, we show how an OWL-S description could be mapped into statechart formalism. Then, we present the statechart model for composition execution in order to achieve the user request. The final section offers concluding remarks and future directions.

1 Hence, in case of non-determinism, we let the user select the appropriate service instance. For example, when several airline companies offer the same flight with the same price, etc.; then the user should select the most suitable one for him.

2. Related work

The literature on Semantic Web services is abundant and the need for a more rigorous formal foundation is widely discussed. Many contributions come from the artificial intelligence community and are often related to classical planning. This similarity to planning problems leads us to use the related techniques for web services composition. A planning problem is generally described by a set of possible states that should be reached, a set of actions that can be used to reach the goal and a set of transitions to reach a state from a given one with a particular action. Hence, a transition is used to describe the preconditions and effects of a particular action. The main difference between web services planning and classical planning problem is the dynamic behavior of web services [14]. The output returned by a service can either be based on input values, or can depend on the internal state of the service and generally it can be obtained in a non-deterministic way. In particular logic-based approaches reduce the problem of checking the existence of a composition into the satisfiability problem in a knowledge base expressed in a Description Logic [16], [18] and [11] or equivalently into the satisfiability of a formula in a theory expressed in a variant of Propositional Dynamic Logic (PDL) [4]. Some approaches have been proposed which use transitions systems as a composition framework with formalisms like Statecharts or Petri nets [13], [2], [15] and [3]. The different approaches are differentiated by the fact that they consider or not non-functional requirements for service composition based on QoS attributes [1] and [12]. These elements allow compensation for composition deficiency and provide criteria for performance measures. These approaches are also differentiated by the fact that they deal or not with the nondeterministic behavior of web services and that the behaviors can not be predicted a priori.

The consideration of user preferences in Web services composition is ignored in many researches. McIlraith and his group [17] propose GologPref system handling user preferences into Web services composition. Their system is based an agent programming language Golog to represent generic procedures and a first-order preference language to represent rich qualitative temporal user preferences. Using these two representations, the system generate Web service composition that realize the generic procedure satisfying the user's hard constraints and optimizing for user's preferences. Authors makes difference between user hard constrains and user preferences, in our work we don't make these difference. We considered that a user can specify his constrains as mono-service constraint related to one service or multi-service constraint related to many services. Other difference concern the language used to express

constraints, they uses a linear temporal logic (LTL) but we use Knowledge Interchange Format (KIF)[2] language with a well defined syntax and semantic. We can express all user constraints types with this language. The verification of constraints is done by checking constraints satisfaction techniques.

3. Composition model

The general architecture of our system "***Semantic Web Services Environment Composition***" (SWSEC) is depicted in Figure 1. As can be seen from this Figure, our system supports the use of existing e-services able to communicate using standard web service languages.

An existing e-service can join existing "service communities" by exporting its service based-community ontology description in OWL-S. Thereafter, the Community Manager Agent (*CMA*) verifies that the imported service *is consistent* with the ontology of that community. In particular, the CMA verifies that the set of operations of the service, their inputs and their outputs are concepts of the ontology of the community; and the preconditions are consistent with those already imported. In addition, the CMA manages how the e-services join or leave the community. It stores the set of OWL-S services descriptions, their localizations in a knowledge base we call the Community Knowledge Base (*CKB*). The interface of a community of services is composed of the set of *common operations* between services composing that community. The Manager Agent (MA) stores the services interfaces of the different communities in the CKB. Thus the concept of community service brings together a set of alternative and potential e-services; and provides the semantic based-ontology of services descriptions. For example, last minute e-bookers or specific airlines provide the common operations of finding, selecting and booking flights by joining the "plane travel" community.

The composition process of e-services is achieved under the collaboration of the "*Composer Agent*" (CoA) and the "*Community Composer Agent*" (CoCA). A CoCA selects a set of available e-services belonging to its community to be instantiated to execute the required operation according to a set of predefined conditions. Hence, its role is to aggregate all responses provided by the different invoked e-services for a process in order to return it to the CoA. The latter is responsible for selecting, coordinating, and assembling the CoCAs in order to execute the required operations. In our system SWSCF, we assume that the request composed with available processes from CKB is *consistent* and it is not underspecified. We also assume that the CoA could interact with the user in case when several instances offer equivalently the same service (or part of it).

[2] logic.stanford.edu/kif/kif.html

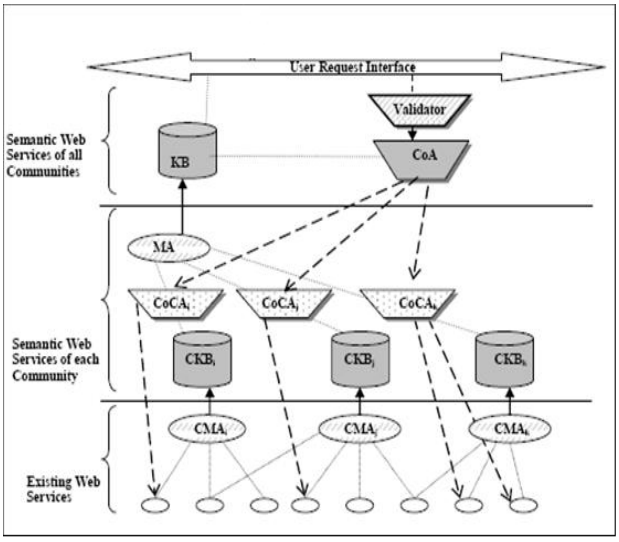

Figure 1. SWSEC general architecture

A user formulates its query using existing available community service descriptions expressed in OWL-S and stored in the CKBs. For the sake of illustration, let us consider the user query described in Figure 2.

A researcher wants to arrange for participation to a conference: he would like to register to a conference, arrange for the travel and book for a hotel room. Therefore, he connects to a publicly accessible repository of services, where he specifies his request and in his turn he receives a service that realizes it (if one exists). Such returned service should interact with the researcher, and allow him to make some choices, which may depend on the results of previously executed interactions.

In general, the researcher request cannot be realized by any single available service, but by a new composite service, obtained by coordinating a set of available services. However the researcher is unaware of how many services are involved in the fulfilment of his request. Consider the situation when the researcher request can be realized by composing the following set of component services: `Conference Registration` service that allows for registering to the conference, `Details Flight` service that gives information in flight travel, `Book Flight` service allows the researcher to book his flight, and `Book Hotel` service for booking a hotel room in conference hotel. Note that a correct execution of the researcher request modelled by the composite `Conference Travel` service consists of all the four operations, performed in sequence order. First, the `Conference Registration` service is executed to make the researcher conference registration, after the `Details Flight` service is executed to get details about plane travel. Based in the previous step result the `Book Flight` service is executed to book flight. At the end step, the `Book Hotel` service is performed to book conference hotel room.

Figure 2 show user query described in OWL-S.

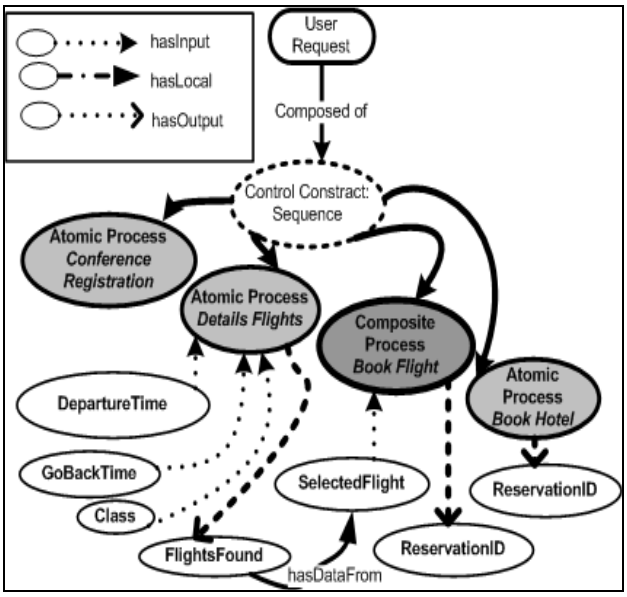

Figure 2. A User request for travel

This representation corresponds to Process Model in OWL-S description. It illustrates user request service control structure (Sequence) and used services. For the sake of readability, service parameters used to express user constraints are presented here. The next section outlines how user constraints are handled by our model during the composition process.

4. Handling user constraints

Although handling user constraints (or preferences) is essential in delivering customized and high-quality services; it has been given little attention in the literature.

User constraints either concern parameters of one single service or many services. In the first case, we call them "mono-service" constraints; whereas, they are called "multi-service" constraints in the second case. In our model, use constraints (mono or multi-service) are modeled as a set of first-order logical formulas related to the inputs, the outputs, and the local parameters of e-services. Hence, we distinguish the following classes of user constraints:

- ***User constraint related to inputs parameters***: it concern restrictions put it in services inputs values. This kind of constraints can be checked before service execution; e.g., flight departure time should be after 08:00 am (DepartureTime >= 08:00 am).

- ***User constraint related to local parameters***: They are checked after the e-service execution; e.g, the plane seat number is between 10 and 20 (seat_number >= 10 and seat_number =< 20).

- ***User constraints related to outputs parameters***: They are checked after the e-service execution; e.g., flight cost less then 500$ (flightCost < 500$).

Obviously, one can get the combination between the three parameters ((inputs, locals), (locals, outputs), (inputs, outputs) and (inputs, locals, outputs)). At this stage, we should remark that a user constraint could be related at once to both input and output parameters. For example, the user wants a *direct flight* costing no more than 650$ (*FlightType* = direct flight and *FlightCost* <= 650$). This kind of constraints is checked in two steps: inputs are checked before e-service execution; and outputs are checked after service e-service execution.

4.1. User constraints descriptions

4.1.1. OWL-S user constraints description. To describe user constraints in the composite service parameters, we add In OWL-S a new property called "hasparameterCondition" used to expression conditions on parameters values. Hence, we define three new properties: hasInputCondition, hasOutputCondition and hasLocalCondition related respectively to input, output and local parameters.

We define these proprieties in OWL-S description as shown in Figure 3.

```
<owl:ObjectProperty
rdf:ID="hasInputCondition">
  <rdfs:subPropertyOf
rdf:resource="#hasInput"/>
  <rdfs:range                    rdf:resource
="&expr;#Condition"/>
</owl:ObjectProperty>

<owl:ObjectProperty
rdf:ID="hasOutputCondition">
  <rdfs:subPropertyOf
rdf:resource="#hasOutput"/>
  <rdfs:range                    rdf:resource
="&expr;#Condition"/>
</owl:ObjectProperty>

<owl:ObjectProperty
rdf:ID="hasLocalCondition">
  <rdfs:subPropertyOf
rdf:resource="#hasLocal"/>
  <rdfs:range                    rdf:resource
="&expr;#Condition"/>
</owl:ObjectProperty>
```

Figure 3. Description of parameters conditions properties.

4.1.2. User constraints expressions language. User constraints, finite set of logical formulas, can be expressed with XML literals languages, such as SWRL [8] or RDF [10]; or with string literals languages as KIF [9] and PDDL [6]. For our concern, we have chosen KIF for the following reasons. First, KIF has *declarative semantics*; i.e., it is possible to understand the meaning of expressions in the language without appeal to an interpreter for manipulating those expressions. Second,

KIF is *logically comprehensive*. Third, KIF permits the introduction of new knowledge representation constructs without changing the language.

To illustrate user constraints description in OWL-S user request service description using KIF language, let consider the example presented in section 2. The researcher has many restrictions related to his trip. These restrictions are expressed as user constraints related to component services. He would like to get an Economic class, the depart time after 08:00 am; and the go back departure time after 07:00 pm. These constraints are related to Details Flight service. Also, he prefers a plane seat between seat number ten and number sixteen; and flight travel and hotel cost less then 800$. These two constraints are about the *Book Flight* and *Book Hotel* services. The full description of user request service with user constrains is shown in Figure 4.

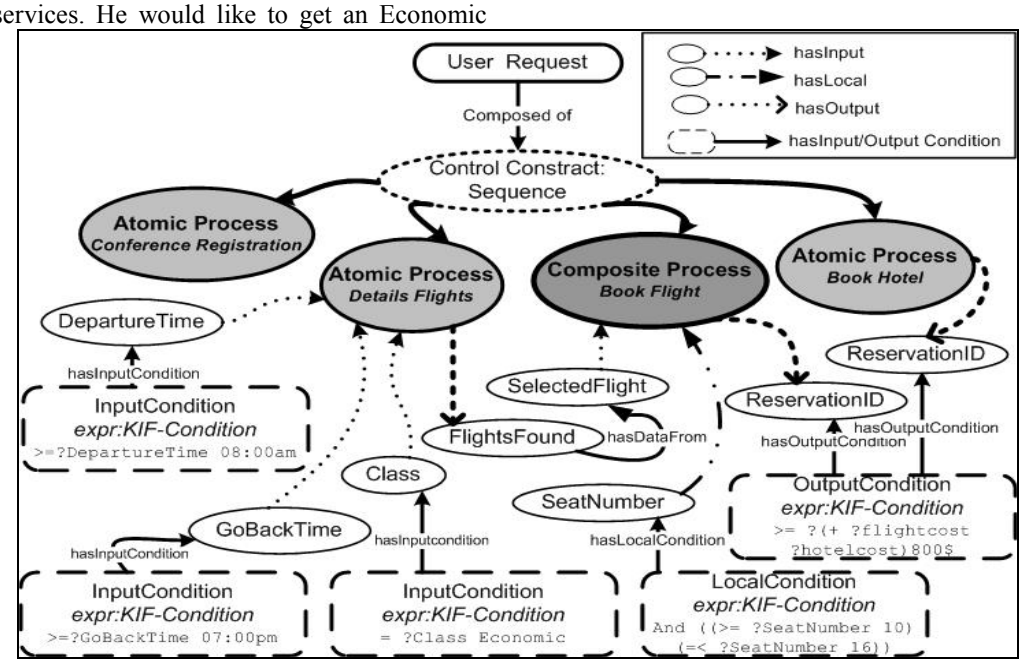

Figure 4 - User request service description handling user requests

4.2. User constraints checking

In our SWSCF framework, user constraints consist of a finite set of constraints whose cardinality may be reduced at each step of the composition process. We denote by $\mathcal{UC}(t)$ the set of user constraints at composition step *t;* and by Card($\mathcal{UC}(t)$), the number of constraints in $\mathcal{UC}(t)$. It is obvious that Card($\mathcal{UC}(t+1)$) ≤ Card($\mathcal{UC}(t)$). This is natural since through composition; user constraints are either completely or partially satisfied, or not satisfied, leading to a reduction in the set of constraints.

User constraints checking were done related to their category type (mon-service or mult-service). For mono-service, we check constraint relatively to service parameters values, but for multi-service constraint, we decompose constraint to mono-service constraints and checking is done partially service by service. In our composition model, user constraints are checked in the execution model by the composer agent (CoA) as we will see in the next section.

5. Execution model

The CoA and the CoCAs collaborate in order to compute an execution strategy of the composition. While the CoA has a *global* view of the execution strategy using "community services", each CoCA has a *local* (or partial) view of the execution strategy since each is concerned only with the offered e-services of its corresponding community. In this section, we will focus ourselves on formalizing the execution model used by the CoA to invoke the CoCAs of the distinct communities. For this, we will use Statecharts [7] as an operational description of the execution model of the composition; mainly due to the fact that it offers the required control-flow constructs and provides an executable model.

OWL-S gives a descriptive representation of services, this description don't present an execution model for services composition. For this it's necessary to map this description to another formalism that present an operational description. This formalism can be Statecharts, Petri nets, process algebra, transaction logic or fine state machine.

The choice of Statecharts as the language for capturing the flow of operation invocations in our model is motivated by several reasons. First, Statecharts possess a formal semantics, which is essential for analysing composite service specifications. Next, Statecharts are a

well-known and well-supported behaviour modelling notation. They are part of the Unified Modeling Language (UML). Furthermore, Statecharts offer most of the control-flow constructs found in existing process description languages: sequence, branching, concurrent threads, and cycles. However, like any other process modelling languages Statecharts have their relative advantages and drawbacks.

In our SWSCF framework all services present an OWL-S description either user query service. These descriptions are mapped to Statecharts representations. We first will describe the *mapping algorithm* from OWL-S to Statecharts formalism. We then present the execution model expressed with Statecharts.

5.1. Mapping from OWL-S description to statecharts model

In OWL-S, processes could be either atomic or composite. They are described by using control constructs as *Sequence, Any-Order, Choice, If-Then-Else, Iterate, While* and *Repeat*. We should notice at this stage that this mapping is *unique*; i.e., for an OWL-S description there is one and only one state chart model.

Each atomic process is represented by a simple state with an initial pseudo state and an end pseudo state. This representation is justified by atomic process semantic (it takes one message in inputs, run on one step and returns an output message). Each composite process is represented by a composite state. If the control construct is sequence, any-order, choice, if-then-else and iterate (without concurrence execution) then the composite service is represented by Or-states statechart. If the control construct is split then the composite service is modeled by And-states statechart. Composite service with split-join control construct is represented by a statechart with And-states followed by join transition. The transition from a service to its successor is modeled by transition in state chart. The data flow between its compounds is modeled by transition actions. The process model parameters are recuperated by actions in statechart. We get service inputs values before service invocation with GetInputs() function related to an ***Entry*** action. Service outputs parameters are returned after service run with ReturnOutputs() function related to ***Exit*** action.

For preconditions parameters, we distinguish two cases: (i) precondition with Local parameters, (ii) precondition without Local parameters. In the first case need a checking step with two possible evaluations. The condition is true, the service state is entered, and otherwise, the system goes to a failed state. For these reasons, we represented precondition with local parameter by a ***Conditional*** pseudo state related to the incoming transition to the state representing the service. In the second case, a ***Guard*** related to the incoming transition to the state is introduced. A Perform service is modeled by a **Do** action (PerformState()). The parameters propriety "InputBinding" correspond to an assign action between two services inputs parameters, are modeled by an ***Action*** (Assign (From; to)) related to the transition between the two states. For a service with an If-Then-Else control construct, its IfCondition propriety is represented by a **Guard** related to the incoming transition to the state representing the service. If a service presents a result expressed by InCondition propriety, the InCondition is modeled by a **Guard** of an internal transition with the event "on Result". The parameters propriety "OutputBinding" correspond to an assign action between to outputs parameters, we represented this propriety in state chart by an **Action** (Assign (From; to)) of an internal transition with the event "on Result". If a service presents an Effect parameter, we modeled this effect by an **Action** (Predicate (term1; term2;…)) of an internal transition. For more details in OWL-S description mapping in statechart, we refer readers to [5].

5.2. Execution model expressed with statecharts

Remind that the main goal is to develop the execution model corresponding to the user request and which will be used by the CoA. This model must take into account: (1) the control flow between operations; (2) the type of service (either providing-information or altering-world); (3) the preconditions of the distinct operations; and (4) the user constraints (either local or global). Each of these requirements is detailed subsequently.

5.2.1. Matching inputs/outputs. Services could be composed if there is a matching between the inputs of the next invoked operation and the current output (which could be either a user predefined input or an operation output). This matching is ontology-based in the sense that it can be reduced to the parameters subsume relations defined in the ontology. The matching task is performed upstream of the state chart composition model before operations' invocations. This task is modeled by a state named `MatchingParameters`. The action performed in this state consists in establishing, for each operation in the order of invocation and for each of its inputs, the parameter to be matched with. In case where this action fails, the CoA cannot go through the next state and the execution model outputs a failure - see Figure 5.

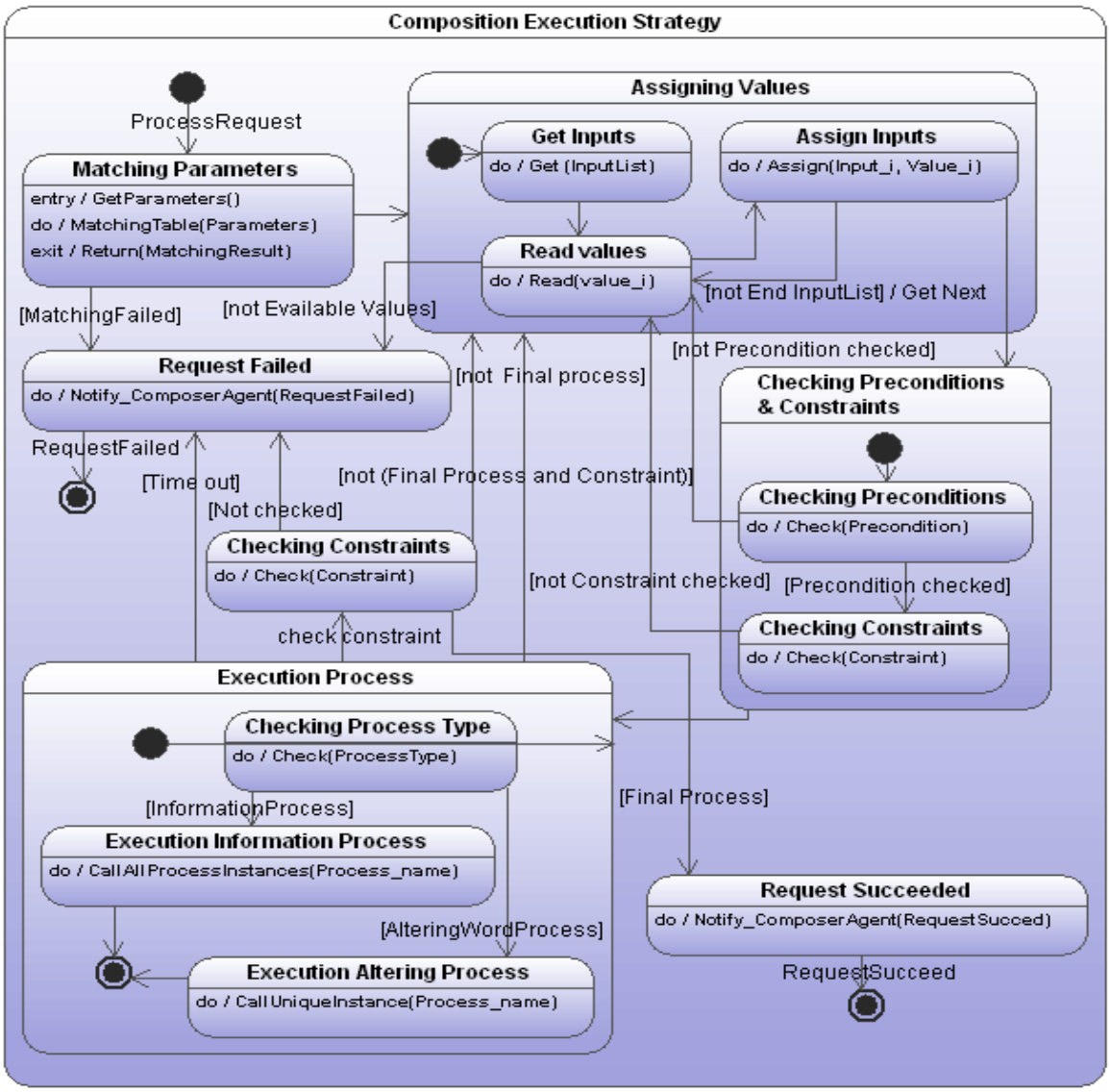

Figure 5. Execution model expressed with statecharts

5.2.2. Non deterministic behaviors. Web services are described by non deterministic behaviors of service processes in the sense that the process outputs (as well as inputs from external processes) cannot be predicted *a priori* execution. For example, several flights complying as yet with some user constraints are found, someone's could not be confirmed by the concerned e-service[3], the CoA must execute the next operation for each value until it finds a value that satisfies all of user constraints. The user request is then executed (`RequestSucceded` state in figure 5). To achieve this, the CoA must be able to perform a backtracking and to invoke the same operation for the next flight.

If all alternatives of inputs are executed and no e-service instance corresponds to the user constraints or processes preconditions, we can say that no instance corresponds to the user request (`Requestfailed` state in figure 5). In the statechart composition model, this task is modeled by a state named `AssigningValues` and located between two successive operations. In this state, the CoA can interact with user to reduce the number of values to be executed. In this example, the user choices only flights that he/she prefers.

5.2.3. Providing and altering-word e-services processes. Besides, once a composer community agent CoCAi is identified and is invoked by the CoA for a given value of each input, it selects using CKBi the e-services offering the required process. Thus, at run-

[3]A flight reservation service cannot know in advance whether a reservation will be confirmed or cancelled by e-service.

time several instances of different e-services simultaneously can be active. This concerns only strictly information-providing e-services. For example, to find a travel CoCAi may invoke different travel e-services at the same time. In other cases, the CoCAi invokes only one e-service instance designed to execute the next process. This is always the case with a world-altering web service processes. For example to book a flight, the CoCAi invokes the e-service which proposed this flight in the previous invocation.

The CoA invocations are the actions performed in `ExecutionProcess` state using `callAllInstances (ProcessName)` and `callUniqueInstance (ProcessName)`. In this state the CoA waits the invocation return until a given timeout is reached in figure 5.

5.2.4. User constraints. In addition to processes preconditions, the CoA must take into account the user constraints which described by KIF language and related to inputs and/or outputs of service. Some ones are local and concern one process at the same time, multi-service constraint concern several processes and could be validated only partially for each process before or after its execution in state `ChekingPrecondition& Constraints`.

Figure 5 depicts the statechart model for composition execution. In this statechart we represent atomic processes sequence. For each control construct, in particular for choice, split-join and split strategies of composition are defined.

5.3. Illustration

To illustrate the execution strategy of an execution model that corresponds to user query, let us consider the example in Figure 4. A researcher wants to arrange for participation to a conference: he would like to register to a conference, arrange for the travel and book for a hotel room.

Consider the situation when the researcher request can be realized by composing the following set of component services: `Conference Registration` service that allows for registering to the conference, `Details Flight` service that gives information in flight travel, `Book Flight` service allows the researcher to book his flight, and `Book Hotel` service for booking a hotel room in conference hotel. Note that a correct execution of the researcher request modelled by the composite `Conference Travel` service consists of all the four operations, performed in sequence order. First, the `Conference Registration` service is executed to make the researcher conference registration, after the `Details Flight` service is executed to get details about plane travel. Based in the previous step result the `Book Flight` service is executed to book flight. At the end step the `Book Hotel` service is performed to book conference hotel room.

The researcher has defined many user constraints restrictions related to his trip. The set $\mathcal{UC}$ of constraints contains four mono-service constraints ($\mathcal{UC1}$, $\mathcal{UC2}$, $\mathcal{UC3}$, $\mathcal{UC4}$) and one multi-service constraint ($\mathcal{UC5}$).

Let $\mathcal{UC} = \{\mathcal{UC1}, \mathcal{UC2}, \mathcal{UC3}, \mathcal{UC4}, \mathcal{UC5}\}$, with:

- $\mathcal{UC1}$: `(>= ?DepartureTime 08:00am)`. The user like that flight Departure time should be after eight o'clock.
- $\mathcal{UC2}$: `(>= ?GoBackTime 07:00pm)`. The user hopes that the flight go back time after seven o'clock.
- $\mathcal{UC3}$: `(= ?Class Economic)`. The user choices an economic class for his flight.
- $\mathcal{UC4}$: `(And ((>= ?SeatNumber 10) (=< ?SeatNumber 16)))`. The user likes a flight seat between 10 and 16.
- $\mathcal{UC5}$: `(=< ?(+ ?FlightCost ?HotelCost) 800$)`. The user likes to get a trip cost less eight hundred dollars.

The user constraints set cardinality is Card($\mathcal{UC}$) = 5. This cardinality will be reduced at each step of service composition by checking constraints. At composition end user constraints set should have cardinality equal to zero.

Our composition model takes three inputs: (i) the set of OWL-S descriptions of services selected for composition, (ii) the user request service description and (iii) the set of user constraints. The model first, starts by mapping step to produce the set of Statecharts describing user request service and, services used in composition. The CoA applied the corresponding strategy, in this case is the sequence composition strategy.

Let us apply our strategy to the example; the first step is the matching parameters between the different services. In our case, the `Conference Registration` service is directly executed because there are no parameters to match, no preconditions to check, and no user constraints related to this service. After, the CoA runs the next service `Details Flight` service. To execute this service, the execution process get the inputs values from the user, each value is assigned to the corresponding input. The inputs are *Departure Airport*, *Arrival Airport*, *OutDate*, *InDate, Class* and *Round Trip*. The values assigned to these inputs are Paris, New York, 03/12/2007, 09/12/2007,

Economic and Ok. Moreover, the user gives departure time value (09:00 am) and Go back time value (07:30pm). After this assignment the execution process passes to Checking Preconditions and user Constraints step. In our case the `Details Flight` service doesn't present preconditions, but there are many user constraints. These constraints are checked after execution for constraints related to outputs or locals parameters, before service execution for inputs constraints. In this case, constraints UC1, UC2 and UC3 are all checked relatively to inputs values (DepartureTime = 09:00am, GoBackTime = 07:30pm and Class = Economic).

After the previous step, the process goes to the Execution Process step. First, it checks the Process Type of `Details Flight` service, it's an information service, and for this the execution process calls all the process instances. The result is a list of flights, checking user constraints, we suppose that is one flight result that corresponds to user constraints, and this result is affected to the output *FlightFounds*. The same steps of strategy are executed for `Book Flight` service. The execution process runs the first step Matching Parameters; *FlightFounds* output is matched to the input *SelectedFlights* of. The result of Matching Parameters step is an "Exact" matching. Then the execution process passes to Assigning Values step, the value of *FlightsFound* output is affected to *SelectedFlights* input. Next, Precondition and user constraints checking step is performed. There is not precondition but there is a user constraints related to Book `Flight` service. UC4 is a constraint related to locals' parameters, it will be checked after service execution. UC5 constraint is a composed constraint for this is partially checked. The plane ticket cost should be less then 800$. The execution process passes to Execution Process step. Before executing the process, it checks his type. `Book Flight` service is an altering world service, for this the executor call one process instance. In our case, we suppose that the flight booked cost 500$ and seat number kept is 12D. The same steps of strategy are executed for `Book Hotel` service that presents user constraints. The executor will check this constraint. The constraint is partially checked in the execution of Book Flight service, it will be completely checked with the execution of `Book Hotel` service. The hotel selected for booking should offers a hotel room cost less or equal to 300$. The executor call one process instance at Process Execution step because `Book Hotel` service is an altering world service.

In this composition run sample, we have supposed that all user constraints have been checked and the outputs values are the desired user values. In the worst case, if one of user constraint is not checked or composition run is failed at a particular step, the CoA interact with user and inform him or it backtrack to take other services parameters values.

In our model, execution space wasn't fully explored at each step. Execution space was reduced by user constraints checking, or by applying heuristics based on services scoring function or by inviting user to choice the value to use.

6. Conclusion

In this paper, we have presented a framework for the semantic composition of web services based on a distributed architecture and able to handle user constraints (or preferences). In this framework, the composition problem is handled at three levels. The first level enables building a description of a composite service which is consistent from semantic point of view. The user personalizes its request by combining the required semantic services descriptions and its constraints using OWL-S expressivity. The second level concerns the composition of semantic descriptions; and the third level concerns the composition of the underlying e-services instances. The responsibility of composing and coordinating the execution of a composite service specified by a user request is distributed across the composer agent (CoA) and the composer community agents (CoCA). The CoA carries out a global view of user request execution using community services descriptions. The CoCA of each community carries out a partial view of user request execution using e-services of its community. This paper focused on modeling the solution that enables describing the execution of composition by the CoA using Statecharts as execution formalism. Our model deals with different aspects of composition problem related to the nondeterministic dynamic behavior of web services and to the fact that the internal states of e-services are unknown *a priori*. It allows the planning tasks to be generated (completely or partially) and updates at run time by interleaving the composition execution by enabling interaction with the user. We can even say that semi-automatic composition could be preferred to fully automated composition because it takes into account the user preferences if several alternatives responding to its request are possible. Moreover, this alleviates the combinatorial problem of composition. Presently, we detail the different composition strategies related to others OWL-S constructs such as choice split-join and split. We plan to study the problem of user request consistency by exploiting the correspondence that exists between OWL and description logic formalism.

7. References

[1] B. Benatallah, M. Dumas, Q. Sheng, "Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services", Distributed and Parallel Databases, Springer Science, 17, 2005, 5-37.

[2] B. Benatallah, M. Dumas, Q. Sheng, A. Ngu, "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services", *in Proc. 18th ICDE IEEE International Conference on Data Engineering*. 2002.

[3] D. Berardi, G. De Giacomo, M. Mecella, D. Calvanese, "Automatic Composition of Process-based Web Services: a Challenge", *in Proc. 14th WWW International World Wide Web Conference*. ACM press, 2005, pp 403-410.

[4] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, M. Mecella, "Automatic Composition of Transition-based Semantic Web Services with Messaging", *The 31th VLDB Very Large DataBases*, 2005, pp 613-624.

[5] Y. Gamha, N. Bennnacer, L. Ben Romdhane, G. Vidal-Naquet, B. Ayeb, "A Statechart-Based Model for the Semantic Composition of Web Services", in Proc. 2007 IEEE Congress on Services, Salt Lake City, 2007.

[6] M. Ghallab et al., "PDDL-The Planning Domain Definition Language V. 2", Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[7] D. Harel, and A. Naamad, "The STATEMATE semantics of statecharts", *ACM Transactions on Software Engineering and Methodology*. vol. 5, no. 4, 1996, pp. 293-333.

[8] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml", 2003.

[9] KIF."Knowledge Interchange Format": Draft proposed American National Standard (dpans). Technical Report 2/98-004, ANS, 1998.

[10] G. Klyne and J. J. Carroll, "Resource description framework (rdf): concepts and abstract syntax", 2004.

[11] S. McIlraith, and T. Son, "Adapting Golog for composition of semantic Web services", in *Proc. 8th International Conference on Principles of Knowledge Representation and Reasoning*, 2002, pp. 195-202.

[12] B. Medjahed, A. Bouguettayan, "A Multilevel Composability Model for Semantic Web Services", *IEEE TKDE*, vol. 17, no. 7, 2005.

[13] S. Narayanan, and S. McIlraith, "Simulation, Verification and Automated Composition of Web Services", in *Proc. 11th International World Wide Web Conference*, ACM Press, 2002, pp. 77-88.

[14] D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu and F. Yaman, "An HTN Planning System". *Journal of Artificial Intelligence Research,* 20, 2003, pp 379-404.

[15] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau and P. Traverso, "Planning and Monitoring Web Service Composition", in *Proc. 2nd ICAPS International Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[16] E. Sirin, P.Bijan, "Planning for semantic web services in Semantic web services", in *Proc. 3rd ISWC International Semantic Web Conference*, 2004.

[17] S. Sohrabi, N. Prokoshyna, and S.A. McIlraith, "Web Service Composition via Generic Procedures and Customizing User Preferences", in *Proc, 5th International Semantic Web Conference*, 2006, pp 597-611.

[18] D. Wu, E. Sirin, P. Bijan, J. Hendler and D. Nau, "Automatic web services composition using SHOP2", in *Proc. Planning for web services workshop in ICAPS*, 2003.

[19] T. Andrews et al. Business Process Execution Language for web Services, http://www.ibm.com/developerworks/library/ws-bpel/

[20] Web Ontology Language for Web Services, http://www.w3.org/Submission/OWL-S/