BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume 7, No 1

Sofia • 2007

An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services[#]

Gennady Agre, Zlatina Marinova***

** Institute of Information Technologies, 1113 Sofia*
*** OntoText Lab. Sirma Group Corp.*
E-mails: agre@iinf.bas.bg zlatina.marinova@sirma.bg

Abstract: *The present paper considers an important problem related to the further development of the semantic Web service technology – the dynamic composition of semantic Web services. It is proposed an approach, in which the process of finding an appropriate service composition is guided by the algorithm for run-time decomposition of the user goal into sub-goals and discovering the existing services able to satisfy these sub-goals. Compatibility of services participating in the composition is achieved by using the consistent description of the composite goal template, prepared by the service application provider in design-time. Using only implicitly provided information about the desired order of execution of services in the composition, the proposed algorithm is able to find a proper orchestration of services in the composition as well as to discover the appropriate service substitutions when some of the services in the composition can not be executed due to some physical reasons.*

Keywords: *Dynamic service composition, Service orchestration, Service choreography, Service discovery, Semantic Web services, INFRAWEBS Framework, WSMO.*

1. Introduction

The desired state for the current software and services is the so called third generation services [12] – these are context-determined, consumer-driven and dynamically composed services. Development of tools, techniques and methods for creating such services will allow for building more flexible service-oriented systems. One of the most challenging problems in designing third generation services is how to compose

[#] The research was partially supported by IIT – BAS, Project No 010079 "Methods and Tools for Processing Semantic Information".

services dynamically, on demand. Dynamic composition requires that existing services are combined in *run-time* (on-the fly) to fulfil the user request, when none of the existing services is able to do this alone. In dynamic composition location of services is based on their capabilities, also used to identify those services that can be combined to create a composition. The full automation of this process is still an ongoing research activity.

This paper presents a data-driven approach to dynamic service composition, developed in the frame of the IST research project INFRAWEBS[1] that was successfully completed in the beginning of 2007. One of the project results is a Semantic Service Engineering Framework, called INFRAWEBS Integration Framework (IIF), enabling creation, maintenance and execution of WSMO-based semantic Web services (SWS), as well as supporting SWS applications within their life-cycle. Being strongly conformant to the current specification of various elements of the WSMO Framework [22], the IIF hides the complexity of creation of WSMO descriptions by identifying different types of users of Semantic Web Service Technology; clearly separating different phases of the Semantic Service Engineering process, and developing a user-oriented software toolset supporting all phases of this process [2].

The basic assumption for the INFRAWEBS model for dynamic service composition is that *this process is realized in the INFRAWEBS IIF environmen*t, which means that:

- all existing services are described as WSMO-based semantic services;
- all services to be used for service composition are published in the P2P-connected INFRAWEBS Repositories (DSWS-R) [17] and can be found by means of the INFRAWEBS Discovery component [19];
- each published service can be executed by means of INFRAWEBS Executor component [24].

In general, semantic service composition is based on using service capabilities, which can be seen as abstract, static descriptions of service functionality. However, in the case of *dynamic* service composition we do not know in advance *what* services need to be composed. In order to avoid this problem we consider *each user request (expressed as WSMO goal) as an abstraction of all services, whose capabilities can match this goal.*

The next step is to assume that if a goal can not be matched to any known service, it can be *decomposed* into several sub-goals, for each of which matching services could be found. Thus, dynamic composition of services can be seen as a process *guided by the dynamic goal decomposition.*

Relating service composition to goal decomposition allows us to make the next constructive step towards an algorithm for dynamic service composition –we assume that *each specific user goal can be seen as an instance of an existing, more general, abstract goal,* known to the system responsible for satisfying user goals. Generally, such an assumption is too strong. However, it perfectly fits the IIF design principles, since in INFRAWEBS dynamic composition is used to serve a special type of users – *Service Application Providers*, who are responsible for creating concrete service applications. In INFRAWEBS, *the functionality (or capability) of a service application* can be fully described by:

[1] http://www.infrawebs.-eu.org

1. *A set of abstract goals (or goal templates),* which, on the one hand express all general requests the user can send to the application, and on the other provide abstractions of all services that can be used to fulfil these goals. Goal templates are prepared by application providers in *design-time* (by means, for example, of such IIF components as SWS-D [1] or SWS-C [6]) and published (stored) in the IIF repository (DSWS-R).

2. A procedure allowing end-user requests to be represented as instances of corresponding abstract goal templates prepared by the service application provider.

The first assumption is still rather demanding to the application provider, since it requires goal templates to be prepared in advance for *all possible* user goals. Being not able to further reduce this demand we, however, are able *to facilitate* the service provider in satisfying it. In order to do this we assume that:

- each goal may be represented either by an *atomic* goal template (representing basic functionality of the application) or by *combination (composition) of atomic goal templates*;
- the IIF provides the application provider with necessary and easy-to-use tools both for goal template creation and for goal combination. Such tools are already available in the IIF – they are SWS-C and SWS-D.

Assuming that the service application capability (represented as described above) is available, the INFRAWEBS approach to dynamic service composition can be sketched as a run-time process consisting of the following steps:

1) representation of the user request as a WSMO goal, based on a particular abstract goal template;

2) decomposing the user goal to a set of sub-goals (if needed);

3) discovering a *compatible* set of existing services – candidates able to satisfy each sub-goal;

4) selecting (done by the user of the application) one concrete service for each sub-goal;

5) execution of the selected services.

In the next sections of the paper we will consider all steps (tasks) of the proposed approach to dynamic service composition in more details. The last section discusses the advantages and drawbacks of the approach and compares it with the related work.

2. Goal decomposition

In order to better understand the methodology for goal decomposition, we firstly present the representation and usage of goals in INFRAWEBS.

2.1. Representation of goals

According to the WSMO specification [24] a goal is represented by:

- A set of imported ontologies used for logical description of the goal;
- A set of logical expressions (WSML axioms) forming the goal capability. Such expressions may have different roles (assumption, precondition, postcondition and effect) describing what the user requires from services able to satisfy the goal, as well as some input data that the user is willing to provide to a service. It is assumed

that only postcondition or/and effect forms the compulsory part of a goal description – other parts of a goal capability are optional;

- A (optional) set of shared variables specifying which variables used in different axioms of the goal capability are global for the whole capability;
- an Interface specifying what interaction the user expects to have with a service able to satisfy the goal. Since the role of this element of the goal description is not fully specified, we do not use it in INFRAWEBS.

As already mentioned, in INFRAWEBS all WSMO goals used in a concrete semantic service application are split to two types:

- *Goal templates* – representing abstract goals describing the application functionality (service application capability). These goal templates are created by the Application Service Provider in design time.
- *User goals* – representing specific request that the user can send to the application. User goals are created in run-time as instances of the corresponding goal templates.

Even though both types of INFRAWEBS goals are represented as WSMO goals, the representation of goal templates is slightly different. In order to implement the mechanism for run-time instantiation of goal templates, it is necessary to mark in advance which variables (parameters) of the goal template are allowed for instantiation (i.e. be replaced by some concrete values) when formulating a concrete user goal. Such "input" variables in the goal templates are marked by extending the list of attributes of a concept, used in the goal description, by a special attribute ("slots") identifying such variables. Marking of such variables is very easy and is graphically implemented in the SWS-D.

A user goal (as instantiated goal template) has the same WSML representation as the corresponding goal template except that some of the input variables (slots) are replaced by concrete values (instances)

Another dimension that should be reflected in a goal description is whether the goal can be decomposed (to some "smaller" sub-goals) or not. In the first case the goal is considered a composite goal, and in the second – an atomic goal. An approach for automatic recognition of the composite nature of a goal via analysis of the logical structure of its capability description is very hard to implement, due to the rather complicated syntax of WSML language [7]. That is why we have applied a significantly simpler implementation approach – in the IIF a composite goal template can be created only by composition of other already created goal templates[2]. This composition can be accomplished by means of such IIF design-time tools as the SWS-C or the SWS-D. In practice, the process of constructing the description of a composite goal is implemented as creation of a new goal, whose capability axioms are built by copying the corresponding axioms from the several goals playing the roles of "sub-goals" of the composite goal under construction. In our implementation such "copy-paste" way of construction of the composite goal is extended by an additional (hidden to the user) operation marking "the source goal", from which the original axiom has been taken. More precisely, each axiom copied from a goal template (source) is pasted to its new (target) place in the capability description of a composite goal along with an automatically created non-functional property (NFP) (dc#relation) pointing to the IRI of the source goal. In this way, all axioms (which can play different roles in the

[2] The composite *user* goal is created automatically in run-time as an instance of the corresponding *composite* goal template.

capability description of the composite goal) having the same value of the NFP dc#relation form a single sub-goal of the composite goal.

Of course, we do not claim that the proposed solution is generally applicable out of the INFRAWEBS context. However, having in mind that there is no generally accepted definition of a composite goal, as well as that the IIF provides easy-to-use tools for graphical, semi-automatic creation of composite goals, we consider the proposed operational definition of a composite WSMO-based goal as appropriate and feasible for our run-time decomposition purposes.

2.2. The algorithm for goal decomposition

The main task of a goal decomposition algorithm is to split the composite goal to a set of sub-goals. If we consider a composite goal as a set of atomic sub-goals $G=\{G_1, ..., G_n\}$, then in general case this task is equivalent to the task of finding all possible subsets of a given set. As it is well known, the number of such subsets is the power set of $G - 2^{|G|}$ ($|G|$ is the size of G).

Even though an approach for finding all possible combination of sub-goals of a given composite goal is feasible for some practical applications that use a small set of atomic goals to be composed, in the general case it does not provide the user with some special advantages. In its simplest variant the decomposition algorithm splits the composite goal G exactly into n atomic sub-goals. For example, such a solution is implemented in the design-time composer prototype GOBAC [3] developed in DIP project.

The main disadvantage of such a solution is the ignorance of existing composite services that, in principle, are able to satisfy more than one of sub-goals at once[3]. The multifunctional character of the INFRAWEBS ideology assumes a special type of IIF user – a service composition provider (or a service broker), who can use the framework (by means of SWS-C) to creating semantic services by means of design-time composition of existing semantic services. In order to allow using such composite services in run-time compositions, we have selected a more complicated solution to the goal decomposition problem formulated above. Informally, the INFRAWEBS goal decomposition algorithm can be described as follows:

- The user goal is sent to the discovery component for finding a set of matching services, each of which is able to satisfy the goal. If such services are found – no decomposition of this goal is done no matter of whether the goal is composite or not.
- If no services have been found to match the goal and the goal is a composite goal, an attempt to decompose the goal is done. The goal is recursively split into two parts (sub-goals) – the first is an atomic goal, and the second is comprised of the rest of the goal under decomposition. The first sub-goal is constructed by all axioms belonging to the composite goal capability having one specific value of dc#relation. The concrete value of this property is obtained from the first axiom found in the postcondition description of the composite goal. If the composite goal has no postconditions (or any postcondition axioms with such non-functional property), the axioms playing the role of effects are used.
- At the next step of the decomposition, both sub-goals are used by the INFRAWEBS discovery component in attempt to find services exactly matching these

[3] More precisely, they *could be used* if a discovery method is equipped with a special mechanism for ranking such composite services higher than "ordinary" atomic services.

sub-goals. Inability to discover any services matching an atomic goal leads to failure of the whole goal decomposition process, while the absence of existing services able to satisfy a composite sub-goal leads to the next step of the recursive decomposition algorithm in which a new attempt to split this composed sub-goal into two "smaller" sub-goals is done.

The implemented algorithm for goal decomposition is a restricted variant of a general algorithm for full combinatorial generation of all possible subsets of a given set. The restriction heuristic of the proposed algorithm may be formulated as "select a possible splitting of a goal to sub-goals according to the order in which atomic sub-goals were used by the application provider in the process of composing that goal". In other words, we assume that application providers usually begin to construct a composite goal with the most important (from their point of view) sub-goal, and continue this process by sequentially adding other sub-goals according to the degree of their importance for realization of the whole goal.

3. Discovery of compatible services

The main problem in the service composition task is to ensure the compatibility of services participating in the composition [4]. In the general case this is done by analyzing the correspondence between the requirements (or constraints) imposed to the services by means of the service orchestration language and the capabilities and interfaces of these services (describing what services can do and how to communicate with them).

As it has been already mentioned earlier, in the dynamic composition of services we do not know in advance what services will participate in the composition. However, in the same way as we have reduced the problem of finding service composition to the problem of goal decomposition, we can narrow the problem of service compatibility to the problem of goal compatibility. In other words, if we can ensure that all sub-goals forming a composite goal are compatible, we will also ensure that all services that exactly match these sub-goals will be compatible too.

The compatibility of semantic services may be considered on two levels:

- Static or functional compatibility, which means that services should describe their functionality by means of compatible ontology terms. In the WSMO framework [25] the general responsibility for such kind of service compatibility falls on ontology-to-ontology mediators (OO-Mediators).
- Dynamic or behavioural compatibility, which means that communication interfaces between services, should be compatible. In the WSMO framework the general responsibility for such kind of service compatibility falls on service-to-service mediators (WW-Mediators).

Currently, mediation is an active field for research in the WSMO community and it was considered too broad area to be tackled in the scope of INFRAWEBS. For this reason, no mediators are supported into the current version of the IIF. Instead, we assume that all services and goals used by a particular application provider are constructed from a common set of ontologies.

However, the problem of static compatibility of services still exists at the level of necessity to provide a mechanism enabling restriction of possible range of different service parameter values, when such services participate in a composition rather than

when used as stand alone services. For example, if a rent-a-car service should be used in a composition with a flight reservation service, usually only rent-a-car services which are located near the airport of the flight destination or are at least in the same city can be used in the composition.

It is clear that the discovery of such compatible services can be achieved automatically if the corresponding restriction is represented as an integrity constraint (or a set of such constraints) in the given composite user goal. Moreover, since in the IIF each user goal is an instance of a goal template, it is enough to put such constraints only in the corresponding composite goal templates. Actually, composite goal templates are a very natural place to do this, since the application provider is the only person who knows what basic functionalities the service application should provide and how to guarantee the consistency of a complex application functionality composed from available services.

Having in mind the concrete form, chosen for representing composite goal templates in INFRAWEBS, we need to define how the application server provider can add such constraints into a composite goal description. Since the representation of all types of goals in the IIF is the same and consists of logical axioms, the general answer to this question is “by modifying the axioms describing the sub-goals of the composite goal”.

This answer is absolutely correct since the goal decomposition algorithm does not require that descriptions of sub-goals produced as a result of the decomposition should be the same as the descriptions of existing goals (or more precisely, goal templates) stored in the application repository. That is why, by appropriate modification of axioms of a composite goal template, it is possible to ensure the compatibility of all services, that will comprise compositions satisfying user goals instantiated from this goal template.

In summary, it is possible to conclude that: the INFRAWEBS approach for discovery of services, which can participate in the dynamic service composition, is based on the use of the consistent description of a composite goal template, created at design-time, which specifies some integrity constraints restricting possible compositions.

4. Selection of services to be used in composition

Service selection is a run-time process in which the user of the application selects a concrete service that should be executed in order to satisfy the user goal. The user selects the desired service from the list of the discovered services, each of which is able (independently) to satisfy the goal. The existing functionality of the Service Access Middleware (SAM) component of the IIF proposes a set of different methods for ranking the service discovery result list according to several criteria which help the user to make more adequate choice [19]. The user selects a service, through the GUI provided by the specific application and then the selection method of SAM is called.

In order to be used effectively during the run-time service composition process, the selection process is extended by two simple functionalities that:

1) Allow the end-user to cancel all services proposed for selection;

2) Keep internal cache of the discovered services as alternatives for the service that has been selected by the user.

The first functionality allows to start the procedure for finding service composition not only in cases when no service satisfying the goal has been discovered, but also when the user prefers to find a composition of new services able to fulfil the goal rather than to use the proposed (composite) service, because of, for example, some "negative" previous experiences with this service.

The second function facilitates run-time service substitution (see next section) and allows reusing of already discovered services if some problems with execution of the selected services occur.

The service selection process results in the creation of a list of services, each of which can satisfy a sub-goal of the composite goal. For each selected service an additional list of alternative services is also cached in order to minimize the time for service substitution in case of failure.

5. Execution of service compositions

Up to now, we have shown how the user goal can be used for finding a set of compatible services, which in combination can (potentially) satisfy this (composite) goal. The evaluation of the overall usability of the discovered services is based only on the static description of service functionalities, represented by their capabilities, and the consistency of the user goal, used to provide integrity constraints for service composition. However, this *unordered set* of services can not be considered yet as a *service composition* because of lack of any information about how the *execution* of these services should be carried out in time (i.e. what is the *orchestration* of these services). In general, this information should be provided by means of an orchestration description provided in the composite goal (which is not yet specified, see the corresponding notes in Section 1). On the other hand, even if the orchestration of goals has been fully specified, in most cases the composite goal designer is unaware of the particular interfaces of services that could match the composed sub-goals. This is due to the fact that services with the same capability might have different interfaces and hence the goal designer cannot be aware of all possibilities. Therefore, we use a different approach – data-driven execution of the service composition, instead of the orchestration-driven one.

We have developed a method for finding the *proper order of execution* of services selected for composition (i.e. a concrete *service orchestration*), which can guarantee the overall satisfaction of the user goal. This method is described in details in the next subsection.

However, even if we are able to properly orchestrate services, there is a possibility that some of them are not executed successfully because of some physical reasons, for example interactive errors, timeout errors, security errors etc. (For classification of different types of errors – see [15]). That is why, in order to complete the description of our approach to dynamic composition of semantic services we have also defined a mechanism for substitution of failing services (see 5.2).

5.1. Orchestrating services

Orchestrating services is the task of finding the proper order of execution of services participating in a dynamic composition. This process depends on the functional

capabilities of the engine that will execute the services (in our case it is the INFRAWEBS SWS-E component [24]). The execution of a WSMO service is guided by its choreography description. The choreography is based on ASM model [23] and consists of a State Signature, describing ontological concepts, used for representing the state of the world in which the service is executed, and Transition Rules, representing how these states are changed. In the IIF the communications between a service application and the SWS-E are carried out through the SAM component, which sends to the SWS-E the IRI of the service to be executed and the current context of the execution. The context is a set of instances of the ontological concepts reflecting the state of the "world" at the current stage of service execution. The result of service execution is sent back to the application through SAM as an updated context (state of the world after the service execution) extended by a status of execution (i.e. whether the execution of the service was successful or there were problems). The choreography engine – the part of the SWS-E responsible for executing the service transition rules - interprets these rules as a monolithic block, which means that it tries to execute the rules until no rules can be fired and executed. It is also assumed that the engine is able to indicate error situations such as impossibility to continue its work because of:

1) the lack of the necessary data in the current state;

2) some physical problems occurring during communication with the Web service (for example the service can not be found in the specified end point or there is no Internet connection with this service at the moment).

While the second case clearly indicates the error situation, the first one should be further analyzed to understand whether the "problem" situation can be "repaired" by asking the user to provide missing information. The SWS-E is enhanced with a mechanism for determining whether user input (in the form of an ontology instance) is missing, based on analysis of the transition rules that cannot be fired. In such case the choreography engine operation is stopped until necessary data is provided by the user. Execution can also be permanently stopped, in which case the corresponding error status meaning "incomplete state of the world" is issued. The decision how to proceed next should be taken by the module for composition execution based on the evaluation of the current situation in which the service is executed.

This module is the composition execution engine (CEE), and the algorithm for recognizing and processing the situations when user input is required can be formulated as follows (see [24] for details):

- if the requested input matches an output concept of some other service choreography, then this value should be acquired as a result of execution of another service;
- otherwise, the user is asked to provide a value for this input concept.

The implementation of this approach has allowed us to reformulate the problem of finding the proper order for execution of a service composition (known in advance before the execution) as a process for dynamic reaching this order according to the following algorithm (inspired by [14]):

1. Create the initial context for service execution from the composite user goal.

2. Try to execute the first service from the composition. If the SWS-E cannot complete the service execution – analyze the situation:

a) if the missing information can be obtained from the outcome of another service – add the "blocked" service to a "waiting list" and continue with Step 4, it execution will continue when this output becomes available;

b) if the missing information cannot be obtained from another service – forward the request to the application user (through SAM) and upon reception, continue execution; the user input is also added to the composition context, in order not to ask the user again, if it is necessary.

3. When service execution is completed, update the composition context by adding the service outputs.

4. Select the next service from the list of services prepared for execution and send it to SWS-E along with the current context.

5. If all services have been executed successfully – check the waiting list:

a) if the list is empty – stop the execution and report "success" of the service composition, else

b) select a service from the waiting list and send it to the SWS-E with the current context,

c) if no service in the waiting list can continue with the current context - stop the execution and report "failure" of the service composition.

6. Repeat the cycle until all services are executed or the composition fails.

5.2. Finding service substitutions

As it has been already mentioned, the problem of necessity to find a proper substitution to a service participating in a service composition, occurs when the initially selected service can not be executed because of some technical reasons, for example interactive errors, timeout errors, security errors etc. It is naturally to assume, that the new service should be equivalent to the "failing" one in the sense that it can fulfil the same sub-goal, which is part of the initial goal that required dynamic composition. According to our approach, the ability of a new service to satisfy the concrete sub-goal will guarantee that the corresponding service will be functionally compatible with the other services already selected as part of the service composition.

The simplest and the most efficient way to find such a service is to use the list of alternative services able to satisfy the same sub-goal. Such matching services have already been discovered and cached in the phase of user goal decomposition. That is why, our approach for finding a substitute of a failing service can be formulated as follows:

1) ask the application user to select a service for substitution from the list of alternative services, previously discovered as able to satisfy the same sub-goal. Report on "substitution failure" if the list is empty;

2) keep the rest of services as alternatives for the selected service;

3) execute the newly selected service instead of the failing one.

6. Implementation

The main steps involved in our approach to dynamic service composition, described in the previous sections, can be logically separated into two tasks: constructing the list of services that together can satisfy the user goal, and executing these services to actually fulfil the goal. In accordance to this separation we have designed the module for dynamic composition (the Run-time Composer) to comprise of two subcomponents:

- service Composition Finder (SCF) – performing goal decomposition and construction of the list of services that together can fulfil the user goal;

- composition Execution Engine (CEE) – responsible for execution of the composed services, as well as for possible error handling.

These two sub-components have been designed to be loosely coupled, in order to enable IIF users to easily substitute one of them if necessary. From the INFRAWEBS architecture point of view, the Run-time Composer is a virtual component because it uses the functionality of other INFRAWEBS components to achieve its task.

As an output, this component provides an execution context which can either provide the user with the results of execution of all the services or specifies that the user goal cannot be fulfilled for some reason. The Run-time Composer functionality is provided as extensions of two other INFRAWEBS components. The SCF provides an extension of discovery (provided by SAM) towards finding a combination of services fulfilling the goal, instead of a single match. Since there is no distinction between composite and atomic goals neither in WSMO nor in the INFRAWEBS approach, no need of changes to the SAM interface was necessary. The SCF functionality is only internally used by SAM in order to enable the user to achieve its goal even if there is no single service that matches.

The Composition Execution Engine, on the other hand, manages the execution of a set of services and was realised as an extension of the SWS-E.

The SCF logic diagram is presented on Fig. 2. Its operation for finding appropriate compositions of services includes:

- Matching of a given goal or sub-goal and providing the list of appropriate services
- Selecting one of the services found as a preferred one to be used for achieving this goal (sib-goal)
- Storing the alternatives as backups in case of service failure
- Recursive decomposition of the current goal into sub-goals in case no matching services were found, or none of the found services was selected.

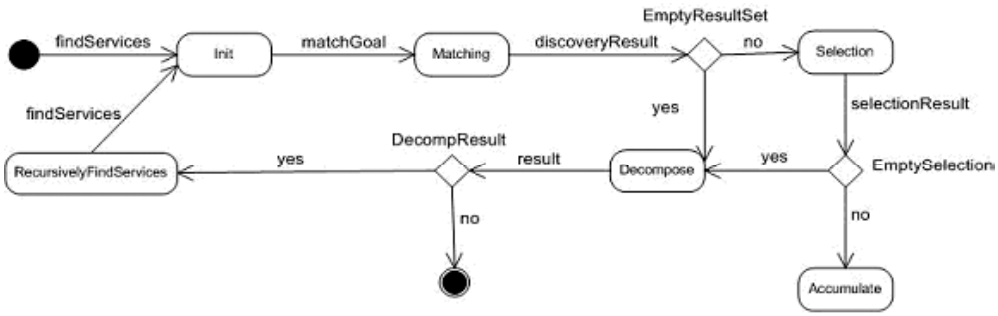

Fig. 2. Service composition finder logic

The CEE is acting as a dynamic orchestration environment able to:

- Coordinate the process of execution of dynamic compositions of services – having no explicit orchestration, but acting in a common context;
- Invoke a service included in the dynamic composition – using the corresponding methods of the SWS-E;
- Support data flow between composed services by means of sharing a common context;
- Provide error handling mechanisms so that alternative compositions can be used in case of service failure.

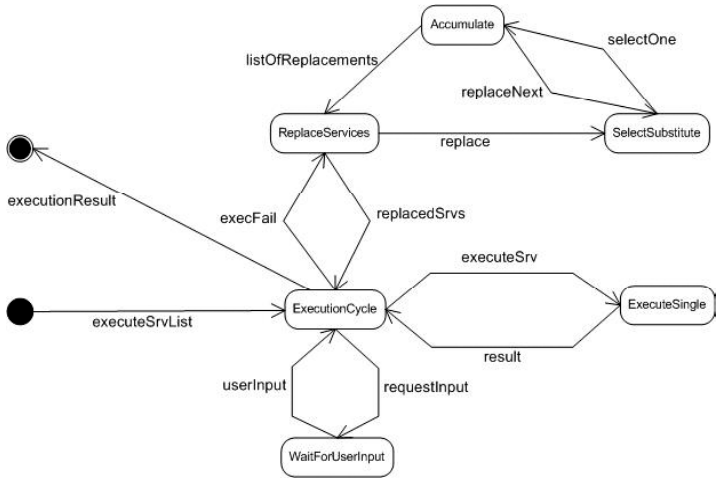

Fig. 3. Composition execution engine logic

Composition execution is initiated by the user, through SAM, and the set of selected services (using the SCF) together with their alternatives is given as input. SAM also creates the necessary Context in which the execution will be carried out. This context is created on the basis of the initial user goal, in the same way in which context is created for single service execution. The logic of work of the Composition Execution Engine is schematically presented on Fig. 3.

7. Discussion and related work

The proposed approach for dynamic service composition was adopted for the preparation of the first prototype application of the INFRAWEBS Framework – the STREAM Flows! System (SFS) [20]. SFS aims at overcoming such shortcomings of existing Frequent Flyers Programs as impossibility of their users to contract services or combination of services using asynchronous, real-time, anywhere and anytime system. The owner of the SFS is STREAM Airlines, which can obtain points from purchasing services of the STREAM group. These services might be an airline ticket, a hotel booking, a car rental and many others. The customer purchases services (paying for them in any kind of money transfer) for many companies (engaged with the SFS program), which collect the information of the customer and send it to the SFS program, adding the counterpart of the service in points of the SFS program. The SFS program collects all the information and stores them into its own databases. The main functionality of the SFS allows the user to create or select travel packages - the choice of a complex package triggers the selection of an appropriate composite goal template allowing the dynamic composition of services (e.g. flight + hotel + car rental) during the execution.

The main advantage of the proposed approach is that it enables finding the smallest possible set of services able, in combination, to satisfy the user goal. The user actively participates in the process of dynamic composition, so that the composition does not involve services the user does not approve of. In this way a quick response time is achieved, as the set of all possible goal decompositions is not

explored. A general drawback is that the decomposition success, as well as the run-time composition success, heavily depends on the initial structure of the user goal, and more precisely on the used goal template. But since in INFRAWEBS goal templates are created by the application provider, it is expected that they have the right structure.

7.1. Scalability and applicability of the approach

The first aspect that has to be discussed is the degree of the dynamism of the proposed approach. In some papers the term "dynamic service composition" is used in the sense that the creation of the service composition is done without any human intervention. In our approach we use the service application user for two tasks – selecting a preferred service to be executed for each sub-goal, and selecting a preferred service for substitution of a service that fails during execution.

Involvement of the user for solving these tasks depends on the concrete application using the IIF, and is not imposed by the approach. It is easy to see, that the human factor can be fully eliminated by replacing the act of the user choice by an automatic procedure for selecting the "best" service based of some criteria, preliminary defined by the service application provider. Such criteria may be the quality of service along with some other parameters, which can be extracted from the user profiles. These criteria may be used to rank the list of discovered or alternative services. The concrete ranking of services in this list can be done by means of different methods provided by the SAM component (for details, see [19]).

Another aspect that has to be discussed is the scalability of the proposed INFRAWEBS approach for semantic service composition, considering the term "scalability" as a possibility to be reused in other (non IIF) environments. From this point of view we consider that our approach is rather scalable since:

- All other approaches assume usage in some specially designed environment providing the proper means for semantic service discovery and execution. From this point of view we also rely on all functionalities provided by the INFRAWEBS Integrated Framework. However, the main functionalities needed for the implementation of the algorithm – service discovery and service execution exist in all other environments relying on the WSMO Framework. Out dynamic composition approach is not bound to any specific implementation of these components. Therefore, the INFRAWEBS Discovery module can be exchanged by the discovery module developed in DIP, or the WSMX execution environment [30] can be used instead of the INFRAWEBS Executor (SWS-E).
- The representation of composite goals used in the INFRAWEBS can also be seen as fully reusable since it completely conforms to the agreed and accepted WSMO specification and does not require any new additions to the "standard" WSML language.
- Moreover, the description of such goals can be constructed by means of other (different from the INFRAWEBS Goal Editor) Editors (for example by the text-based Goal Editor of WSMO Studio [28] or by the corresponding editor from the Web Service Modeling Toolkit (WSMT) [16]).

The last aspect to be discussed is the applicability of the INFRAWESB Run-time Composer, having in mind the amount of restrictions the approach imposes on services to be used in the service composition.

- The most restrictive assumption of the proposed approach is the use of a common set of ontologies for describing both the goals and services. However, this

restriction is caused by the current status of the work on ontology mediators in WSMO community and will be easily relaxed in the future.

- Another drawback of our approach is the impossibility to explicitly specify the desired control flow for execution of services in the composition. However, this deficiency of the approach can be avoided by more effective use of the goal interface description, when the goal orchestration language is fully specified.
- The current implementation of the Composition Execution Engine invokes services in a sequential order. However, this is not a restriction of the algorithm but an implementation decision based on the current implementation of the Executor component. The principle of the blackboard architecture, which is intensively used by the algorithm, allows parallel execution and synchronization of services. Of course, it is assumed that there is no interference between the services participating in the composition.

7.2. Related work

Service composition is part of a more general research area - automated workflow composition, which is a field of intense activity, with applications in at least two wide areas: Business Process Modelling and (Semantic) Web Services. A detailed overview of methods for automated workflow composition can be found in [5]. Tentative techniques to address this problem are experimented using many formalisms and techniques, among which: situation calculus [18], Logic programming [25], type matching: [9], coloured Petri nets [29; 10], Linear logic [21], Process solving methods [6; 13; 27], AI Planning [8], Hierarchical Task Network (HTN) planning [26], and Markov decision processes [11].

It is interesting to see how the same problem is solved in other approaches for service composition. For example, a similarly implemented approach for WSMO-based service composition (but in the design time) is the one developed in the DIP Integrated project [3]. There, service composition is considered as a goal-oriented process, which at a glance can be described as follows:

1) the user defines his request to the composer in the form of a composition goal involving atomic goals plus constraints;

2) atomic goals present in the request are used to discover matching semantic Web services;

3) the choreographies of matching SWS are used as input to the composer, as elementary bricks of the solution;

4) the composition goal is fed to the composer as constraints that restrict the possible constructions;

5) the composer calls a configurator to produce a valid orchestration from the available choreographies under the specified constraints;

6) the result is then exported as a SWS description involving a valid orchestration description, in the WSML-AD extension.

As it can be seen, the service selection step is not explicitly mentioned in this schema, however its results are explicitly used at the step 3, when the choreographies of selected (not of all matched) services are used for producing a valid orchestration.

Another example is a prototype service composer, which is based on the OWL-S paradigm for representing semantic services [26]. In this approach called “composition-driven filtering and selection of semantic Web services”, “the service

composition requires a human in the loop. A human who has the domain knowledge for the task should guide the overall composition process where the composer aids the operator by providing the relevant choices at each step". In other words, the user operates explicitly with services rather then with goals. She composes the chain of services one by one, using the system for filtering services which are not compatible with the description of the services already selected to be part of a composition. It is clear that such a process can be seen neither as a run-time composition nor as a dynamic one and, it is more appropriate only for static, design-time composition of services.

7.3. Conclusion

The INFRAWEBS approach to dynamic composition of WSMO-based services composition can be summarized as follows:

- The process of finding an appropriate service composition is guided by the algorithm for run-time decomposition of the user goal into sub-goals and discovering the existing services able to satisfy these sub-goals
- Compatibility of services participating in the composition is achieved by using the consistent description of the composite goal template, prepared by the service application provider in design-time.
- Compensation of the lack of special control and flow constructs in the service orchestration language by an algorithm for data-driven execution of the composition of services and means for exchanging data between services, which are fully based on the existing specification of WSMO services and goals.

The proposed approach for dynamic service composition has been implemented as the INFRAWEBS Run-time Composer - a conceptual sub-component of the IIF run-time environment. Using only implicitly provided information about the desired order of execution of services in the composition, the Run-time Composer is able to find a proper orchestration of services in the composition as well as to discover the appropriate service substitutions when some of the services in the composition can not be executed.

The Run-time Composer is implemented as two separate sub-modules – Service Composition Finder and Composition Execution Engine. The first one is integrated with INFRAWEBS Service Access Middleware (SAM) component of the IIF, and the second – with INFRAWEBS Execution component. Both components are accessible as Web services.

References

1. Agre, G. INFRAWEBS Designer – A Graphical Tool for Designing Semantic Web Services. – In: AIMSA 2006, LNAI 4183, Springer Verlag Berlin Heidelberg, 2006, 275-289.
2. Agre, G., T. Pariente, Z. Marinova, H-J. Nern, A. Lopez, A. Micsik, A. Boyanov, J. Saarela, T. Atanasova, J. Scicluna, O. Lopez, E. Tzafestas, I. Dilov. INFRAWEBS – A Framework for Semantic Service Engineering. – In: E. di Nitto, A-M. Sassen, P. Traverso, A. Zwegers (Eds.). At Your Service: An Overview of Results of Projects in the Field of Service Engineering of the IST Programme. MIT Press Series on Information Systems, 2007 (in press).

3. Albert, P., C. de Sainte Marie, L. Henocque, M. Kleiner. Goal-oriented SWS composition prototype. DIP Deliverable 4.22, December 14, 2006.
4. Budak Arpinar, I., B. Aleman-Meza, R. Zhang, A. Maduko. Ontology-Driven Web Services Composition Platform. – In: Proc. of IEEE International Conference on E-Commerce Technology (CEC'2004), San Diego, USA, 2004.
5. Atanasova, T., H. Daskalova, V. Grigorova, D. Gulev. Design & Realisation of Case-Based Composition of SW Services in Design Time (Design-time Composer). INFRAWEBS Deliverable D5.4.2, September 2006.
6. Benjamins, V. R., D. Fensel. Special Issue on Problem-Solving Methods. – International J. of Human-Computer Studies (IJHCS), **49**, 1998, No 4, 305–313.
7. Bruijn, J., H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, D. Fensel. D16.1 – The Web Services Modeling Language (WSML). WSML Draft, October 2006.
8. Carman, M., L. Serafini, P. Traverso. Web Service Composition As Planning. – In: Proc. of ICAPS03 International Conference on Automated Planning and Scheduling, Trento, Italy, 9-13 June, 2003.
9. Constantinescu, I., B. Faltings, W. Binder. Large Scale, Type-Compatible Service Composition. – In: Proc. of IEEE International Conference on Web Services (ICWS 2004), San Diego, USA, 2004.
10. Dijkman, R., M. Dumas. Service-Oriented Design: A Multi-Viewpoint Approach, CTIT technical report series No 04-09. Technical report, Centre for Telematics and Information Technology, University of Twente, Netherlands, February 2004.
11. Doshi, P., R. Goodwin, R. Akkiraju, K. Verma. Dynamic Workflow Composition Using Markov Decision Processes. – International J. of Web Services Research, 2, January-March 2005, No 1, 1-17.
12. Fitzgerald, B., C. Olsson (Eds.) The Software and Service Challenge. Contribution to the preparation of the Technology Pillar on "Service, Grid, Security and Dependability" of the 7th Framework Programme, Ver. 1.1., 30 January 2006.
13. Gomez-Perez, A., R. Gonzalez-Cabero, Manuel Lamaa. A framework for Design And Composition Of Semantic Web Services. – In: Semantic Web Services, 2004 AAAI Spring Symposium Series, 22-24 March, 2004.
14. Hunt, J. Blackboard Architectures. JayDee Technology Ltd., 2002.
15. Kavantzas, N., D. Burdett, G. Rizinger, T. Fletcher, Y. Lafon. Web Service Choreography Description Language Version 1.0, 2004. **http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/**
16. Kerrigan, M. Developers Tool Working Group Status. Version 1, Revision 4. **http://wiki.wsmx.org/index.php?title=Developer_Tools.**
17. Marinova, Z., G. Agre, D. Ognyanov. Final Dynamic DSWS-R and Integration in the IIF. INFRAWEBS Deliverable D4.4.3, February 2007.
18. McIlraith, S., T. Son. Adapting Golog for Composition Of Semantic Web Services. – In: Proc. of Conference on Knowledge Representation and Reasoning, April 2002.
19. Micsik, A., L. Kovacs, Z. Toth, P. Pallinger, J. Scicluna. Revise after final Specification & Realisation of the Discovery Component. INFRAWEBS Deliverable D6.2.2. August 2006.
20. Pariente Lobo, T., A. Lopez Perez, J. Arnaiz Paradinaz. Demonstrator. INFRAWEBS Deliverable D10.8.3, February 2007.
21. Rao, J., P. Kungas, M. Matskin. Logic-Based Web Service Composition: From Service Description to Process Model. – In: Proc. of 2004 IEEE International Conference on Web Services, ICWS 2004, San Diego, California, USA, July 6-9 2004.
22. Roman, D., U. Keller, H. Lausen (Eds.) Web Service Modeling Ontology, 2006, WSMO Final Draft.
23. Scicluna, J., A. Polleres, Dumitru Roman, D. Fensel. D14v0.2. Ontology-Based Choreography and Orchestration of WSMO Services. WSMO Final Draft 3 February 2006.
24. Scicluna, J., Z. Marinova, G. Agre. D7.4.3 Final SWS-E and Running P2P-Agent. INFRAWEBS Deliverable D7.4.3, February 2007.
25. Sirin, E., J. Hendler, B. Parsia. Semi Automatic Composition Of Web Services Using Semantic Descriptions. – In: Proc. of ICEIS-2003 Workshop on Web Services: Modeling, Architecture and Infrastructure, Angers, France, April 2003.

26. S i r i n, E., B. P a r s i a, D. W u, J. H e n d l e r, D. N a u. HTN Planning for Web Service Composition Using SHOP2. – J. of Web Semantics, **1**, 2004, No 4, 377-396.
27. T h a k k a r, S., C. A. K n o b l o c k, J. L. A m b i t e, C. S h a h a b i. Dynamically Composing Web Services From On-Line Sources. – In: Proc. of AAAI-02 Workshop on Intelligent Service Integration, Edmondon, Canada, July 2002.
28. **http://www.wsmo.org/**
Last accessed May 2007.
29. Y i, X., K. K o c h u t. A Cp-Nets-Based Design And Verification Framework For Web Services Composition. – In: Proc. of 2004 IEEE International Conference on Web Services, July 2004, San Diego, California, USA, July 6-9, 2004.
30. Z a r e m b a, M., M. M o r a n, T. H a s e l w a n t e r, H o-K y u n g L e e S u n g-K o o k H a n. D13.4v0.3 WSMX Architecture. WSMX Working Draft 12-10-2005.