Automated and flexible composition based on abstract services for a better adaptation to user intentions

Emna Fki, Saïd Tazi, Khalil Drira

Automated and flexible composition based on abstract services for a better adaptation to user intentions

Emna Fki, Said Tazi, Khalil Drira

CNRS, LAAS, 7 avenue du colonel Roche F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France

Abstract

In recent years, the composition of loosely coupled servies with the aim of satisfying the user intention is a widely followed research topic. The composition of services implies the ability to select, coordinate, interact, and interoperate existing services. This is considered as a complex task. This complexity is mainly due to the large number of available services and their heterogeneity as they are created by different organizations. This complexity is increased when services must be dynamically and automatically composed to meet requirements which are not satisfied by existing services. In fact, an approach for service composition must offer the potential to achieve flexible and adaptable applications, by selecting and combining services based of the request and the context of the user. In this perspective, different approaches have been developed for services composition. However, most of the existing composition approaches tend to be static and not flexible in the sense that they do not have the ability to adapt to user requirements.

To overcome these challenges, we propose a composition approach in which the generation of the composition schema is performed at runtime through the use of abstract services provided at design time. The composition process that we propose takes as input a structure of user requirements materialized by a graph of intentions and enriches this graph to explicit the implicit relationships. The enriched graph is used to generate an initial composition schema by building the control flow and selecting the appropriate abstract services. The selection of these services is based on the semantic matching and the degree of semantic affinity between abstract services. Then, the final composition schema is generated using a refinement mechanism of abstract services using semantic matching techniques and taking into account user context and constraints.

Keywords: service composition, selection, abstract service, user intentions

Email address: efki@laas.fr,tazi@laas.fr,drira@laas.fr (Emna Fki, Said Tazi, Khalil Drira)

1. Introduction

The last few decades have been marked by the rapid development of distributed information systems, and the spread of Internet access. This evolution has led to the development of new paradigms for interaction between applications. One of these paradigms which has grown considerably in recent years is service-oriented architecture (SOA). The design approach of SOA is based on standards which enable creating an integrated IT infrastructure capable of rapidly responding to new user needs. Actually, it is not always easy to find services that meet user requests. Therefore, the service composition satisfying the user intention is a growing need. The composition of services implies the ability to select, coordinate, interact, and interoperate existing services. The composition is considered as a complex task. This complexity is mainly due to the large number of available services and their heterogeneity as they are created by different organizations [1]. This complexity is increased when services must be dynamically and automatically composed to meet requirements which are not satisfied by individual services. In fact, an approach for service composition must offer the potential to achieve flexible and adaptable applications, by selecting and combining services based of the request and the context of the user. In this perspective, different approaches have been developed for services composition. However, most of the existing composition approaches tend to be static and not flexible in the sense that they do not have the ability to adapt to user requirements.

To build a composition of services, two steps must be performed (separately or combined) [2]:(1) a composition schema (or a process model) specifying the control and data flows between activities must be created; (2) concrete services have to be discovered and assigned to activities of the process. As regards the degree of dynamicity in these two steps, we have retained two strategies of composition. The first consists in defining the composition schema at design time and selecting concrete services at run-time based on automatically analyzable criteria, such as QoS parameters. The second strategy consists in combining the generation of composition schema with the selection of concrete services at run-time. This implies that the composition of services may be performed completely at run-time. Generally, artificial intelligence (AI) inspired methods are used in order to provide a fully automated composition. An example of a fully automated composition of services based on planning algorithms is given in [3]. To create a composite service, the service requester only needs to specify the initial and final states for the composite service. Then, the generation of the plan can be achieved by using an expert system based on rules. Although this second strategy seems to be more flexible and adaptable than the first one, modeling the control and data flows of a composite service is potentially a tedious and a time consuming task in terms of reasoning and planning, and especially if we consider a fully automated composition. In addition, the creation of data flow is complex and may require user intervention.

In the work presented in this paper, we try to take advantage of the benefits of both strategies mentioned above: in addition to the selection of concrete ser-

vices at run time, the generation of the composition schema is partly performed at run-time using abstract services provided at design time. This enables flexibility and adaptability without having to build a composition of services from scratch. We argue that many of the required tasks are repeated frequently, and the main steps to achieve these tasks are well known, at least at an abstract level. Nevertheless, the realization of such tasks varies as it is adapted to individual users. As such, our proposal is to specify such tasks using a set of abstract services being able to present alternative services and to combine and refine them with user constraints at run-time. We propose an approach which is composed mainly of four steps. The first step takes as input user requirements materialized in a graph of intentions. This graph is enriched by implicit relationships between intentions. The result of this step allows to generate an initial composition schema by constructing the control flow inferred from the enriched graph of intentions and selecting the adequate abstract services. The choice of these services is based on semantic matching and the degree of affinity between abstract services. The third step consists in generating the final composition schema using a refinement mechanism of abstract services based on semantic matching. Finally, the execution plan is generated taking into account the non-functional constraints provided by the specification of intentions. We have used the OWL-S [4] ontology, to define our services at the abstract service level. In this paper we are focusing on the phase of generating the composition schema based on abstract services. To perform this phase, we have defined mechanisms of semantic matching at different levels as we will explain later.

The remainder of the paper is organized as follows: section 2 introduces some motivating illustrations to show treated issues. Section 3 provides an overview of the proposed composition approach. We present in section 4 the preliminaries by introducing the specification of abstract services and the model of user intentions. In section 5, the proposed automated composition process is described, and specifically the step of composition schema generation will be treated in details. Section 6 resumes composition steps by applying them on a case study. In section 7 we evaluate our approach through experimental results. An illustration of some existing works is shown in section 8. Finally, we provide concluding remarks.

2. Motivating illustrations

Motivating illustration 1 - Problem of static and predefined composition schema. As a motivating example, consider the case of the management of energy in smart buildings. Let us assume that the process achieving the optimizing of energy consumption must be established. This process needs to take into account the fact that the user have to work in the most comfortable conditions. In order to achieve this goal, a series of tasks need to be conducted such as regulating luminosity and regulating temperature. For each task, there may be a great number of sub-tasks and a variety of relationships between sub-tasks. This can be due to the difference of infrastructure and equipment of each Building, the type of rooms,etc. For example, in the temperature regulation phase there

are many alternatives: we can make action on heaters, or on air-conditioners, or on both, depending of the availability and state of these equipment. The adjustment of the heater for example shall take into account if the room is a manipulation room. It should also consider the temperature degree that guaranties the comfort of the user.

It is not feasible to predefine every case in the composition schema due to the inherent complexity. This calls for dynamic generation of composite service schemas based on some rules and context information.

As far as we know, most existing composition approaches assume that composition schemas are static and predefined. This mode of composition needs the establishment, a priori, of all task relationships. It also needs meticulous transformation of relationships and rules into a particular composition schema. If different alternatives to achieve a goal can be found, those alternatives must be enumerated. The optimal strategy can be selected at execution time. This way, a huge number of execution paths is predefined. Thus, adaptation and evolving the process at runtime become more difficult. In addition, it is not easy to predefine composition schemas in complicated application domains where processes might have to be adapted to specific users. Consequently, there is a need to enable automated and dynamic generation of composition schemas adapted to user needs.

Motivating illustration 2 - Waste of time and energy in the concrete service discovery and selection. As a second example, let consider a user looking for a travel service. The user might have to invest considerable resources to visit numerous sites, to determine adequate service providers, to provide his preferences, to integrate or align the different types of results coming from the different sites.

With the increasing number of Web services available within over the Internet, locating the suitable Web service with respect to user's requirements is challenging. To discover and to rank services online may not be suitable, especially from the performance point of view in the context of Internet-scale environment. Instead, concrete services may need to be assembled and generalized within a structure featuring an abstract description of the common functionality offline. Therefore, services to be selected online with respect to user's needs should be limited within one or several abstract services, and then the task of locating a desired service should be appropriate to be performed online.

Motivating illustration 3 - Inappropriate composite services with respect to user intention. Let consider the same user (from *Motivating illustration 2*) who looks for a travel service. The inputs of this service are duration and destination. The outputs are travel packages including expense and travel schedule. For the same request, a service composition system could construct a conference search service because of the similarities of inputs and outputs. We need to keep in mind that the request doesn't specify preconditions and effects. The result does not match the user intention.

Figure 1 illustrates how inappropriate composite services can be generated.

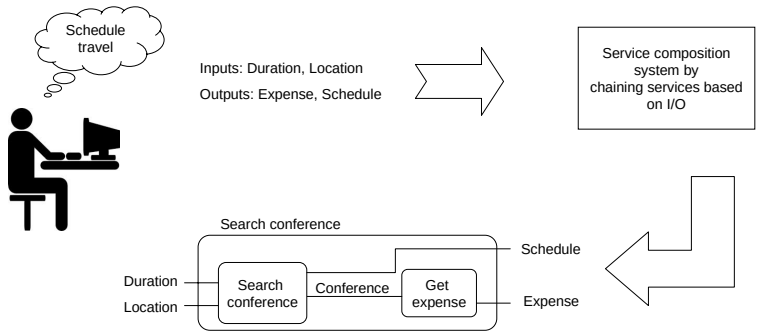

Figure 1: Example of service composition considering only service inputs and outputs

3. Overview of our composition approach

In this section, we provide an overview of our composition approach which deals with the issues presented in the previous section.

The proposed service composition approach involves four consecutive steps: intentions graph enrichment, generation of the initial composition schema, generation of final composition schema, and generation of execution plan as shown in Figure 2.

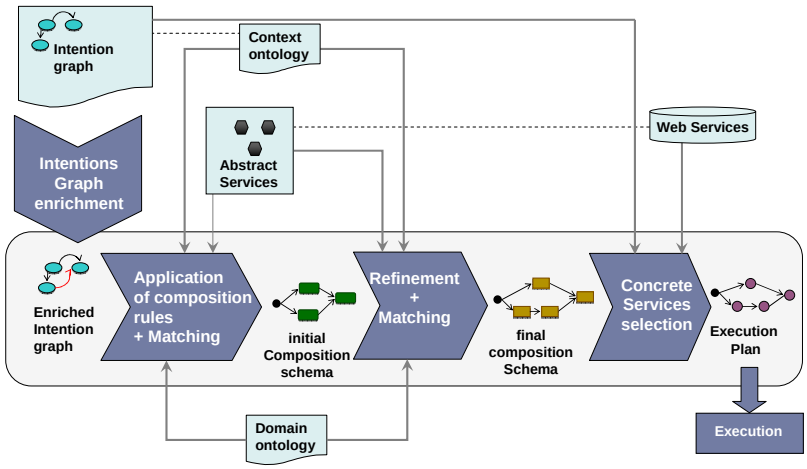

Figure 2: Service composition steps

The composition process takes as input a graph representing intentions specification. We have defined a set of enrichment rules. The application of these rules transforms intentions specification into an enriched graph of intentions. Then, this latter is used to generate an initial composition schema by applying composition rules and by matching intentions to appropriate abstract services.

This matching is done through semantic rules exploiting a domain ontology and uses the context ontology to find abstract services best suited to contextual situations. The initial composition schema is refined by selecting other abstract services of finer granularity until obtaining a final composition schema constituted of atomic processes. The concrete Web services that can fulfill the obtained composition schema are selected according to required non-functional constraints. Thus, the execution plan is generated. As mentioned before, the composition schema represents a high level service process model. This model defines only the set of activities and the control flow among them. It does not identify the concrete services to be invoked. The execution plan is a materialized composition schema in which appropriate actual services are selected and assigned to each involving task.

In this paper, we will focus mainly on the generation of the composition schema.

4. Preliminaries

Before detailing the composition steps, we first explain the basic concepts and definitions to be used later. We need specifically to introduce and specify abstract services and to formulate user intentions.

4.1. Abstract Service

4.1.1. Presentation and motivation

One of the important research issues that influences the way services are composed is service description. Especially, functional descriptions of services are needed in the composition schema planning phase, while QoS descriptions are needed in the selection phase. In our approach, we use abstract services which feature a generic and reusable description. An abstract service is defined over concrete services and presents the typical ways of composing services to achieve particular goals. This would decrease the high cost of discovery and selection tasks of the actual Web services. Since the same abstract service could be offered to several circumstances, different service compositions derived from this service can be obtained at different levels of granularity ranging from the functional properties of services to the execution related properties. The main motivation behind abstract service proposition are the following issues: (1) The need to facilitate matching between client intentions and available concrete services, (2) The need for flexibility in composing services to satisfy particular intentions, and (3) The importance of reusability of services.

Abstract services are composed to construct complex processes. They constitute the design-time components in our approach. They are pre-defined by an expert of a particular domain. The expert knowledge is key for the design of abstract services. However, these services are set to evolve dynamically through learning methods or case based reasoning for example.

An abstract service generalizes the commonalities of a group of concrete services and provides a certain level of abstraction over these services. We

use the OWL-S language to represent and implement the proposed abstract services. The OWL-S language provides mechanisms of abstraction to represent processes. Furthermore, it offers semantic information that enables automated service discovery and combination.

Several frameworks have been developed on top of the OWL-S language to support Web service composition. However, the majority of the Web services specified with OWL-S are connected to concrete Web services. Consequently, the composition reasoning process is principally focused on concrete Web services, thereby having to deal with huge search spaces. An abstract service may have different implementations and multiple ways of combining services of lower granularity. Whereas an OWL-S concrete service has only one composition process. Furthermore, it maps to only one concrete web service at runtime.

4.1.2. Specification of abstract services

The ontology of an abstract service is mainly composed of a Profile part and a Process part. The abstract service is described by its profile which provides the information needed to describe the service functionality, including the functionality of the service and the set of inputs and outputs. Our approach, similar to [5], describes the service functionality as an action and an object. For instance, a service which ensures a hotel reservation has a functionality that can be specified as {reserve, hotel}.

The *process* part specifies the typical composition of the Web services. It defines either a class of services that the abstract service is abstracted from or a generalized workflow that achieves the abstract service functionality (cf. Figure 3). In the *process* part of our Abstract OWL-S service, a service is modeled as a process that can be either composite process or atomic process. In the case of a composite process, the abstract service is composed of other sub-processes which are simple processes. Their composition can be specified by using one of the control constructs such as *Sequence* and *If-Then-Else*. This latter control construct offers alternatives to satisfy the functionality of the abstract service. The introduction of the variability in the design of the conditional abstract service is motivated by the need to introduce flexibility in achieving the functionality, and adaptability in the process of composition of services. A composite process of an abstract service is composed of simple processes. A simple process can be seen as an abstract view of an atomic or composite process. Simple processes cannot be invoked and are not associated with a Grounding. If the abstract service process is atomic, the service is associated with one or more concrete OWL-S services through the relationship *implementableBy* (cf. Figure 3). The implementation of each concrete OWL-S service and the details of how to access the actual web service are specified by the *Grounding* part of the concrete OWL-S service. This set of concrete OWL-S services constitutes a collection of web services having the same functionality and having different non-functional properties (eg, different suppliers, different QoS values, etc.). These non-functional properties are specified in the profile part of each concrete service. At run-time, the appropriate candidate service is selected and invoked. For environments and frameworks that require adaptation or self-repair

at run-time, all candidate services can prepare services for the replacement or substitution.

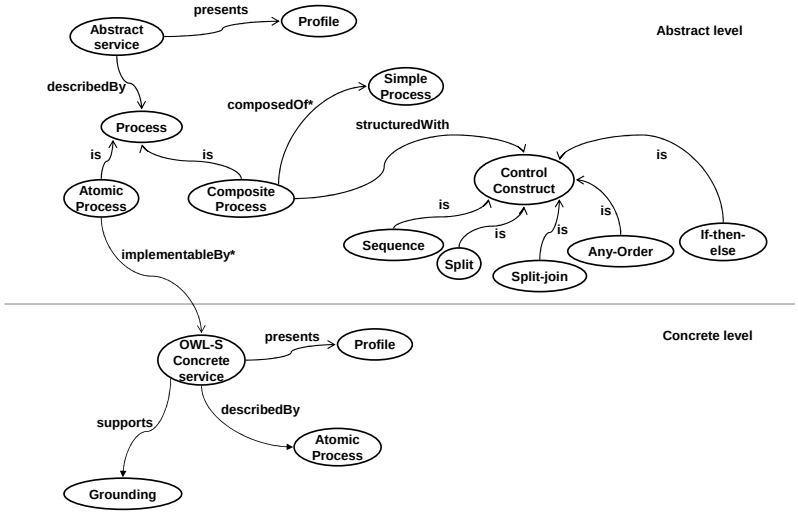

Figure 3: Abstract service model

4.2. User intention representation

To benefit from the user needs which guide the provision and composition of services, we define the concept of intention. This concept is a combination of a goal and a set of constraints expressing the way in which this goal is accomplished (cf. Figure 4). These constraints constitute a set of functional and non functional constraints. As an example, for a goal such as *Plan travel*, there can be a functional constraint like *travel fees <2000$*. Non-functional constraints can be related to execution constraints such as the execution time of a service. These constraints can be also related to security. As an example, a service requiring authentication could be requested by the user. The *goal* of an intention is represented by the pair: action and object (of the action). For example, the action of *Plan travel* is *Plan* and its object is *travel*. We have identified two structural relations by means of which intentions are related. We have been based, for this, on the theory of Grosz and Sidner [6]. These authors have identified two structural relations between intentions, fundamental for the analysis of the structure of the discourse at a basic level: the relation of dominance and the relation of satisfaction precedence. An intention I_1 dominates an intention I_2 if the satisfaction of I_2 contributes to that of I_1. Intention I_1 precedes (the satisfaction of) I_2 if I_1 must be satisfied before I_2.

Each intention specification is modeled as a directed attributed graph GI = <I,R> where: I represents the set of intentions, and R represents the relations between the intentions. I=<G,C> where G: represents the

goal of the intention, and C represents the constraints expressed by user. C = <FC,NFC> where: FC = {fc_1,fc_2,...} is the set of functional constraints, and NFC = {nfc_1,nfc_2,...} is the set of non-functional constraints. R=<d,p>; where d represents the dominance relation and p represents the precedence relation. Each intention is associated with one or more context parameters. Figure 4 presents the semantic model of the intentions that defines the different relations between concepts.

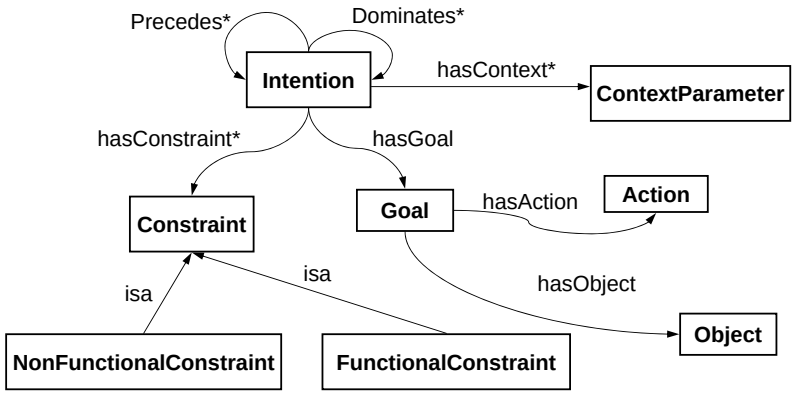

Figure 4: Intention ontology model

In this work, our concern is not extraction of intentions. In the literature, some research studies [7] focus on this issue: intention specifications are extracted from user requests or context information. We assume that the intention specifications are available and we can use it.

5. Composition of services

Since we are focusing, in this paper, on the generation of the composition schema based on abstract services, we will simply use "service" to refer to abstract service in the rest of the paper.

5.1. intentions graph enrichment

The goal of this step is to make explicit the implicit precedence relationships between leaf nodes of the graph.

The composition process starts by receiving a graph of intentions as input (ex: the graph G_0 in Figure 5).

We assume that the graph of the original intentions is complete and includes all necessary information for enrichment. This graph is enriched by applying two rules which state that the precedence relation between two intentions A and B is propagated to intentions dominated by A and B respectively. This makes

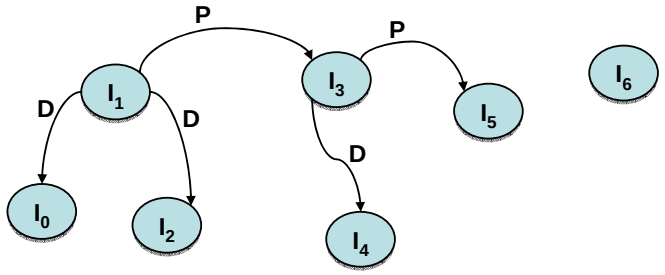

Figure 5: G_0: original graph of intentions

Table 1: R1: Rule1 for the intentions graph enrichment

$Intention(?x) \wedge Intention(?y) \wedge Intention(?z)$ $\wedge precedes(?x, ?y) \wedge dominates(?x, ?z) \rightarrow precedes(?z, ?y)$

possible to identify the relationship between intentions that do not dominate other intentions (leaf nodes). We have defined the following enrichment rules:

Rule R1 (cf. Table 1) identifies an intention x that precedes an intention y and intention z dominated by x, then adds a precedence relation between z and y.

Table 2: R2: Rule2 for the intentions graph enrichment

$Intention(?x) \wedge Intention(?y) \wedge Intention(?z)$ $\wedge precedes(?x, ?y) \wedge dominates(?y, ?z) \rightarrow precedes(?x, ?z)$

Rule R2 (cf. Table 2) identifies an intention x that precedes an intention y and an intention z dominated by y, then adds a precedence relation between x and z.

The application of rules R1 (cf. Figure 6) and R2 (cf. Figure 7) leads to the enrichment of the graph G_0 by new precedence links:

- links (I_2-I_3), (I_0-I_3) and (I_1-I_4) by identifying the link (I_1-I_3)
- link (I_4-I_5) by identifying the link (I_3-I_5)
- link (I_2-I_4) by identifying the new precedence link (I_2-I_3)
- link (I_0-I_4) by identifying the new precedence link (I_0-I_3)

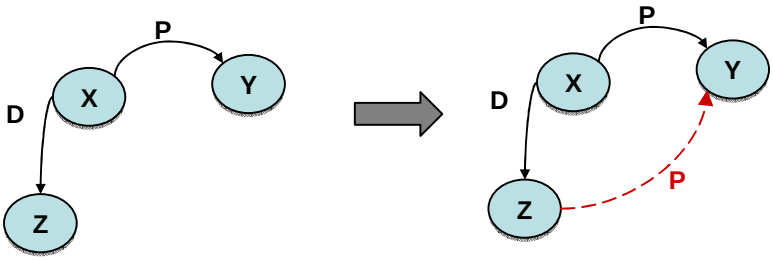

Figure 6: Modification of intentions graph by R1

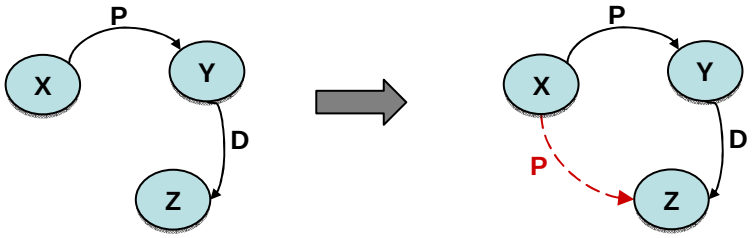

Figure 7: Modification of intentions graph by R2

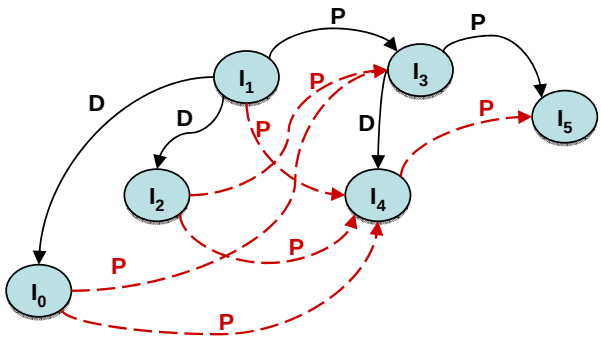

Figure 8: Enriched intentions graph (G_1)

We get finally an enriched graph G_1 (cf. Figure 8) making explicit the implicit precedence links of G_0.

Figure 8 shows an example of an enriched intention graph. Dashed links are those which are deducted from the enrichment operation.

5.2. Generation of initial composition schema

In this step, the adequate abstract services that usable in the composed result are identified and selected. The specific order in which they are composed is inferred automatically from dependencies expressed in the intentions specification. Therefore, we have here two steps: building the control flow and the selection of abstract services as shown in Figure 9.

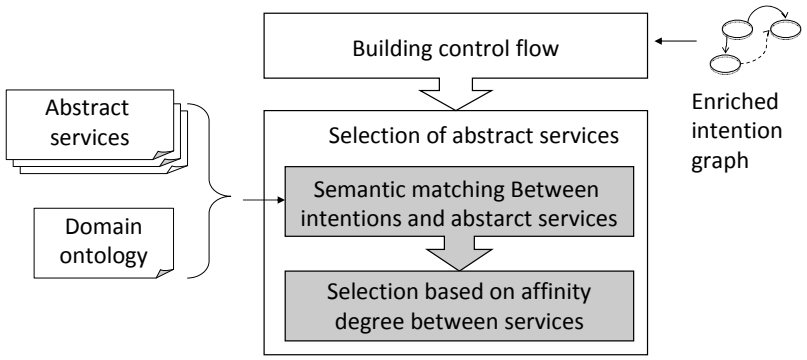

Figure 9: Steps of initial composition schema generation

5.2.1. Building control flow

The control flow refers to the order in which the services must be ranged through a composition. To identify the control flow, we have defined the four rules that act on the intentions graph as follows:

- The precedence relationship which can be deduced by transitivity are deleted.
- Only intentions which do not dominate other intentions are mapped to appropriate abstract services. This eliminates redundancy.
- The precedence relation is translated into a sequence relationship between corresponding abstract services: the execution of the service corresponding to the preceded intention requires performing of the service that precedes it.
- The independence between intentions is translated into a parallel relationship between the corresponding services.

5.2.2. Selection of abstract services

The phase of abstract services selection that constitutes the initial composition schema is composed of two steps depicted in Figure 9.

For the selection of abstract services that match the intentions, we rely on a semantic approach in order to find the most appropriate services. Our matching algorithm compares semantically user intentions with abstract services.

Step1: semantic matching between intention and abstract service. In order to find a set of appropriate candidate abstract services that meet an intention, we calculate the semantic matching degree between the intention goal of the service functionality name. This matching is based on the use of ontologies and the degree of semantic similarity. In our work, we calculate a score of matching between the intention and the abstract service using the following formula: $score(matching) = score(matching\_action) + score(matching\_object)$

As we mentioned before, the goal of an intention consists of an action and an object. Thus, to determine the matching score between an intention and an abstract service, we calculate: (1) matching score between the action of the intention (action_I) and the action of the abstract service (action_S), and (2) matching score between the object of the intention (object_I) and the object of the abstract service (object_S).

This matching is based on the use of an action ontology (for action matching) and a domain specific ontology (for object matching). The action ontology defines action concepts which can be provided through a domain-specific ontology of actions or through general-purpose ontology that includes all the possible action (such as Wordnet). Levels of matching between actions are based on four relation types: Exact, Synonym, Hyponym, and Hypernym. Each level of matching is associated to a score, as defined in [8]: Exact:1, Synonym:0.9, Hyponym:0.7, Hypernym:0.6.

A domain-specific ontology represents possible objects in a specific domain. It is used to perform the matching between objects by using the subsumption hierarchy between concepts. Paolucci et al. in [9] have identified four levels of matching between two ontology concepts: Exact, Plug-in, Subsume, Fail.

The score of matching between object_I and object_S is determined using the function $d(C_i, C_j)$ which measures the semantic distance between two concepts, C_i and C_j. The work of Rada et al. [10] describes the semantic similarity between two concepts of an ontology as follows:

$S(C_i, C_j) = \frac{1}{d(C_i, C_j)+1}$

We are not interested in subsumes matches as we consider that this degree of match cannot guarantee that the intention will be satisfied by the service: we opt to select only the services whose functionality is equivalent or more generic than the intent of the user, in order to avoid the case where the service does not perform the required functionality. Therefore, the matching between objects is based on two types of semantic matching: exact and plug-in. In this case, if $object_I \equiv object_S$ (exact matching), then $d = 0$; if $object_I \subseteq object_S$ (plug-in matching), then $d = 1$.

Services having a total matching score that exceeds a threshold (specified by the designer) are added to the set S of candidate services.

Step2: Selection based on affinity degree between services. After finding candidate service sets, all different alternatives for the initial composition schema are extracted.

For example, for the intention I_2, two services s_2 and s'_2 could be selected. For the intention I_4, two services s_4 and s'_4 could be selected. Each of intentions I_5 and I_6 matches only one service: s_5 and s_6. For this case, we obtain four alternatives for the initial composition schema as shown in Figure 10.

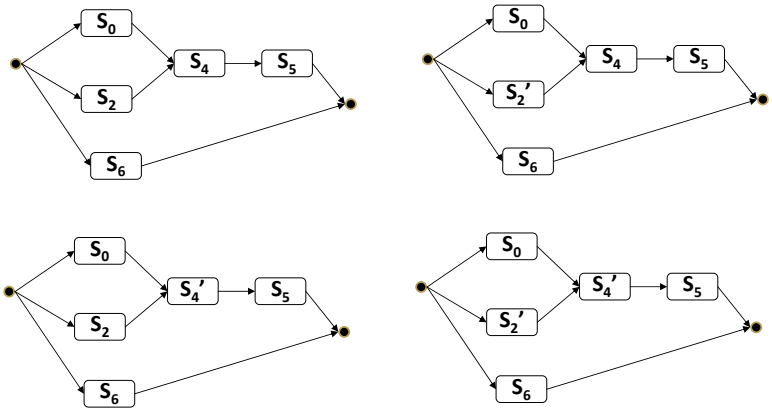

Figure 10: Different alternatives of the initial composition schema

The selection of the adequate alternative is based on the semantic connection quality between services (relation between input and output parameters). These

Table 3: Semantic matching functions described by *Sim*

Match type	*Exact*	*Plug-in*	*Subsume*	*Fail*
$Sim(out, in)$	1	$\frac{2}{3}$	$\frac{1}{3}$	0
Semantic meaning	$out \equiv in$	$out \subset in$	$out \supset in$	Otherwise

parameters are concepts of an ontology. Thus, it is a question of calculating the degree of semantic similarity between output parameters $s_x.Out$ of service s_x and input parameters $s_y.In$ of service s_y, with s_x and s_y being two services composed sequentially.

Thereby, s_x and s_y are semantically linked according to a matching function $Sim(out, in)$, with $out \in s_x.Out$ and $in \in s_y.In$.

The four types of semantic matching functions proposed by [9] are considered to check semantic similarity between a concept *out* and a concept *in*. The semantic similarity is valued by the function Sim (cf. Table 3) to determine the semantic degree of link between parameters of services.

The *Plug-in* match means that an output parameter of a service s_x is subsumed by an input parameter of the succeeding service s_y whereas the *Subsume* match means that an output parameter of service s_x subsumes an input parameter of the succeeding service s_y.

In our approach, the valuation of the semantic similarity in this step is not used to chain services; otherwise, we do not need to find which services depend on other services. Indeed, the sequence relationship is already established, that is that there is at least one pair from input parameters of s_y and from output parameters of s_x having a semantic dependency relation. In other words:

$\exists out_j \in s_x.Out, \exists in_i \in s_y.In$, such that $Sim(out_j, in_i) > 0$

Our goal here is to find the best composition schema according to an optimization criteria. This criteria is the quality of semantic connection between services. Thus, we consider the matching degree between input and output parameters of services to preserve those offering the maximum values of semantic similarity.

Let $s_x.Out = \{out_1, out_2, ..., out_n\}$ be the output set of s_x, and $s_y.In = \{in_1, in_2, ..., in_m\}$ the inputs set of s_y.

In order to determine semantic similarity between s_x and s_y, it is necessary to evaluate the connection between $s_x.Out$ and $s_y.In$. We measure the matching value between each input parameter in_i of s_y and each output parameter out_j of s_x in order to finally retain the maximum value. The pair (out_j, in_i) that has the maximum similarity value represents the couple for which the value of out_j is consumed by in_i. Then the sum of these maximum values is calculated and divided by the number of inputs of s_y. The semantic affinity which represents

the dependency degree between s_x and s_y is given by the function *semAff*:

$$semAff(s_x.Out, s_y.In) = \frac{1}{m}\sum_{i=1}^{m}\max_{j=1}^{n} Sim(out_j, in_i) \quad (1)$$

with $(0 \leq semAff(s_x.Out, s_y.In) \leq 1)$

It is worth reminding that *semAff* function (1) is used to calculate the degree of semantic affinity between only two sequential services. In practice, it is possible that a service s_y is preceded by more than one service; we should therefore take into account all of the outputs of services preceding s_y. Let consider the set of parallel services S that precedes s_y. We need to find for each input parameter of s_y a output parameter from the set of output parameters of services contained in S, with which the similarity is maximum. Figure 11 shows an example of correspondence between outputs of services in S and input parameters of s_y. The input in_3 of s_y, for example, has the maximum similarity with the output out_2 of S.

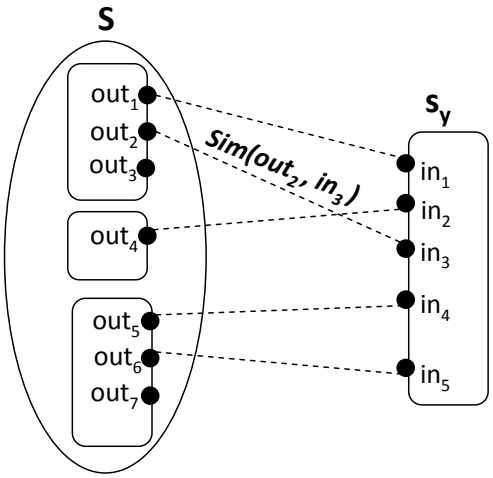

Figure 11: Correspondence between S outputs and s_y inputs

Indeed, it is unknown a priori which service of S that provides the output parameter that aliments an input parameter of s_y. The algorithm 1 finds out pairs *(output parameter, input parameter)* corresponding to the maximum similarity value, and therefore identifies for every service of S, the set of output parameters that will be consumed by the input parameters of s_y. This finally allows to calculate the semantic affinity between each service of S and the service s_y.

Let consider $s_y.In$ the set of input parameters of s_y and $S.Out$ the set of output parameters of all services included in S.

Input: s_y,S
Output:
foreach *input* $in_i \in s_y.In$ **do**
  $Sim_{max} \leftarrow 0$
  foreach *output* $out_j \in S.Out$ **do**
    if $Sim(out_j, in_i) > Sim_{max}$ **then**
      $Sim_{max} \leftarrow Sim(out_j, in_i)$
      $out_{max} \leftarrow out_j$
    end
  end
  foreach *service* $s_k \in S$ **do**
    if $out_{max} \in s_k.Out$ **then**
      $sum_{maxSim[s_k]} \leftarrow sum_{maxSim[s_k]} + Sim_{max}$
      $nbInput_k \leftarrow nbInput_k + 1$
      //number of inputs of s_y *to match with outputs of* s_k
      $\{Inputs_k\} \leftarrow \{Inputs_k\} \cup in_i$
    end
  end
end
foreach *service* $s_k \in S$ **do**
  $semAff(s_k, s_y) \leftarrow sum_{maxSim[s_k]}/nbInput_k$
end

Algorithm 1: Algorithm of semantic affinity calculation between sequential services

The step of semantic matching between an intention and a service can give two candidate services s_k and $s_{k\prime}$, which have same possibilities of matching for a set of input parameters of s_y: $\{Inputs_k\}$. So the semantic affinity between $s_{k\prime}$ and s_y is calculated as:

$$semAff(s_{k\prime}, s_y) = \frac{1}{nbInput_k} \sum_{i=1}^{nbInput_k} \max_{j=1}^{p} Sim(out_j, in_i)$$

with $\{out_1, ..., out_p\}$: the set of outputs of $s_{k\prime}$ to match with the set $\{Inputs_k\}$ of inputs of s_y.

The next step is to identify the best matches in order to find out the best initial composition schema. The best composition schema is obtained by maximizing the sum of semantic affinity values of pairs of sequential services as follows:

$$maximize \sum semAff(s_i, s_j)$$

with s_i et s_j two sequential services.

Figure 12 presents an example that shows different possible alternatives to obtain composition schema. Considering this example, the value of the semantic affinity between s_{01} and s_{41} (case (1)) has the highest value compared to other alternatives ((2), (3) and (4)). However, alternative (1) is discarded, and alternative(2) is retained as the sum of the semantic affinity values $semAff(s_{01}, s_{42}) + semAff(s_{42}, s_5) + semAff(s_2, s_{42}) = 0.8 + 0.7 + 0.6 = 2.1$ is the maximum value.

Thus, services s_{01} and s_{42} are the selected services.

What remains now is to establish the data flow between selected services, ie to link compatible inputs/outputs.

5.3. Generation of final composition schema

In this step, the initial composition schema will be further refined: more abstract services of finer granularity will be selected iteratively until obtaining a final composition schema formed by atomic abstract services.

During this refinement, the *process* part (cf. Figure 3) of each service composing the initial composition schema is analyzed. We distinguish two cases:

(i) In case of atomic *process*, no further refinement is required. The set of candidate concrete OWL-S services that implement this process will be analyzed during the phase of the generation of the execution plan.

(ii) In case of composite *process*, each of its sub-processes (which are all simple processes) is analyzed: for each simple process, a search is performed to find corresponding abstract services.

A simple process is substituted by the *process* part of the corresponding service. The corresponding service is that which matches simple process according to a degree of similarity. If the *process* part of this service is an atomic process, the procedure described in (i) is applied. If the *process* part is composite, this

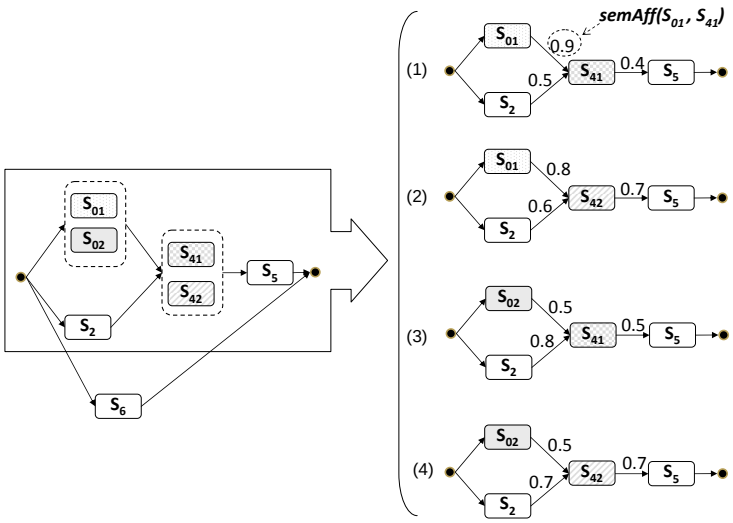

Figure 12: Example of possible alternatives of initial composition schema

simple process is extended to a composite one. And hence, we will be situated in case (ii).

The substitution requires that the substituting service has a functionality equivalent to the functionality provided by the original simple process. The control flow of the composition schema is updated by the refinement process. In fact, the structure of the composite processes that feed iteratively the composition schema is expressed by the control construct defined in the abstract service (cf. Figure 3). Some of these control structures impose a specific order of the activities of the *process* such as *Any-Order* and *Sequence*. Other control structures are used for the choice of one activity from a set of activities such as *If-then-else* and *Choice*. In fact, some abstract services propose alternatives to achieve the functionality of the service. Each alternative is influenced by a specific functional constraint or contextual data. During the refinement, appropriate sub-process is selected. The information provided by the specification of intentions and context are used to guide the selection of adequate activities.

We rely on the concept of similarity that we used in subsection 5.2. We evaluate the degree of similarity between the simple process and the corresponding service. To determine the degree of similarity between the sub-process SPr and the service s, we calculate the semantic distance between:

(i) the sub-process name and the service name.

(ii) Input and output parameters of the sub-process and the service.

The condition (C) allowing a service s to be considered as possible match for a sub-process SPr is the existence of input and output parameters of s whose semantic distance with input and output parameters of SPr is not null on one

hand. On the other hand, the semantic distance between names of the service s and of the sub-process SPr must be non null. This condition can be expressed as follows:

$$C \equiv \begin{cases} \forall in_k^{SPr} \in SPr.In, \exists in_p^s \in s.In \\ \text{ such that } d(in_k^s, in_p^{SPr}) > 0; \\ \forall out_k^{SPr} \in SPr.Out, \exists out_p^s \in s.Out \\ \text{ such that } d(out_k^s, out_p^{SPr}) > 0; \\ d(s.Object, SPr.Object) > 0; \\ d(s.Action, SPr.Action) > 0; \end{cases}$$

with $SPr.In$ and $s.In$: sets of input parameters, $SPr.Out$ and $s.Out$ sets of output parameters, $s.Action$ and $s.Object$ constitute the name of the service, and $SPr.Action$ and $SPr.Object$ constitute the name of the sub-process.

Let $\mathbb{S}$ be the set of services s_i that satisfy the condition C for a sub-process SPr. We look over $\mathbb{S}$ by calculating the semantic distance D between s_i and SPr using the following formula:

$D(SPr, s_i) = \sum_{j=1}^{|SPr.In|} d(in_j^{SPr}, in_j^s) + \sum_{j=1}^{|SPr.Out|} d(out_j^{SPr}, out_j^s) + d(s.Object, SPr.Object) + d(s.Action, SPr.Action)$

The set $\mathbb{S}$ is updated, it contains henceforth services S_k that have the minimum distance.

The selection of services is also conditioned by elements of the context attached to the description of the intentions. Conditional services offer the possibility to evaluate a condition in order to choose the adequate alternative (sub-process). The condition is usually associated with a context item, such as the existence of presence sensors in a building. In addition, we take into account the functional constraints, especially in the last level of refinement which provides a composition schema composed of processes of atomic services. For example, a functional constraint is about the measurement unit of a device which gives the temperature.

5.4. Generation of execution plan

Once the final composition schema is generated, the next step of composition process consists in identifying concrete services which will constitute the execution plan. This step is based on non-functional constraints provided by the specification of intentions in order to select the appropriate services. Recall that an atomic abstract service is associated with a set of OWL-S concrete services. The implementation of each concrete OWL-S service and details of how to access the service and invoke it are specified in the *Grounding* part. This set of concrete services represents a collection of web services having a common functionality having different non-functional properties (ex, different providers, different QoS values, etc.). These non-functional properties are specified in the *profile* part of each concrete service. As we mentioned, a set of concrete services candidates is associated with each abstract service. This association is done at design time and can be updated by service providers. This particular issue is not addressed in this work.

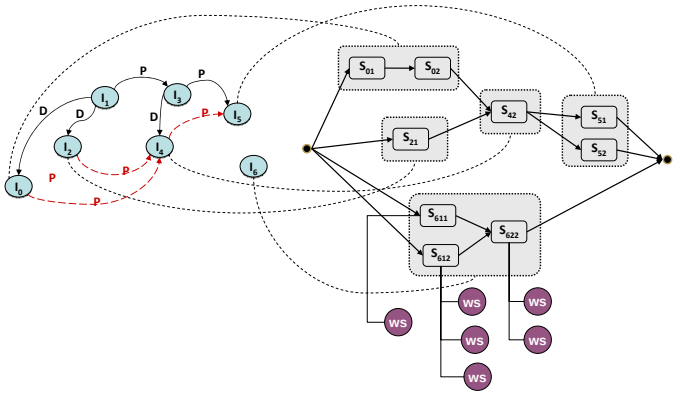

Figure 13: Non-functional constraints propagation

It is necessary to select the concrete service that allows to implement the abstract service taking into account the non-functional constraints defined in the intentions graph, as shown in Figure 13.

We would again point out that the abstract service that is mapped to an intention I during the phase of the generation of the initial composition schema could be refined during the phase of the final composition schema generation. This would give birth to a combination of a set of services $\mathbb{S}_{\mathbb{I}}$ representing a fragment of process performing the intention I. Therefore, it is not a matter of locally considering the best concrete service (taking into account the QoS criteria) which can realize each abstract service of $\mathbb{S}_{\mathbb{I}}$. In other words, all the concrete services that will be selected for the abstract services of $\mathbb{S}_{\mathbb{I}}$ must satisfy non-functional constraint(s) expressed in I to obtain the best fragment of process corresponding to I. It is therefore necessary to propagate the non-functional constraint of the intention I to all concrete services which carry it out because the refinement process could give a fragment of process that performs a single intention.

If we consider for instance the execution time criteria, we should calculate the total execution time of the fragment of process corresponding to a given intention by the aggregation of execution time values of services composing this fragment. More generally, the aggregation function depends on the QoS attribute and on the control flow of the composition. We present in Table 4 aggregation functions for attributes: execution time, cost and availability. q_T, q_P, q_D denote the values of QoS attributes: execution time, cost and availability. The set of services s_i represents the concrete services corresponding to the n abstract services which constitute the composition (whose structure is either sequential or parallel).

Let us take the case of the intention I_6 which has the non-functional constraint: *execution time* $<$ *x secondes*. The intention I_6 is achieved by a process

Table 4: Aggregation functions of QoS attributes

QoS attribute	Control structure of the composition	
	Sequence	Parallel
Execution time	$\sum_{i=1}^{n} q_T(s_i)$	$\max_{i=1}^{n}\{q_T(s_i)\}$
Cost	$\sum_{i=1}^{n} q_P(s_i)$	$\sum_{i=1}^{n} q_P(s_i)$
Availability	$\prod_{i=1}^{n} q_D(s_i)$	$\prod_{i=1}^{n} q_D(s_i)$

fragment containing three services (cf. Figure 13): s_{611}, s_{612} and s_{622}. The execution time of this fragment should not exceed x seconds, thus we must have:

max(Execution time (s_{611}), Execution time (s_{612}))+ Execution time (s_{622}) < x secondes.

5.5. Implementation

To validate our proposed approach, we have implemented the composition process. The prototype was implemented in Java using the environment Eclipse with free tools and open sources. We used the Pellet[1] reasoner to infer implicit relationships of the intentions graph, and to calculate similarity measure. We used *Protégé* editor (version 3.2.1) to manipulate ontologies. To create and edit abstract services described with OWL-S language, we used specifically the OWL-S Editor tool implemented as an open source plugin under *Protégé* editor. Several Java APIs were used to handle ontlogies and OWL-S services like OWL API and OWL-S API[2].

Our prototype includes the following main modules:

- Module of intentions processing: the input of the module is an OWL file describing the specification of intentions of a user. This module applies enrichment rules over the graph of intentions.
- Module for building control flow: it builds the composition schema without taking into account abstract services that will constitute it. In other terms, it generates its overall control structure. It takes as input all non-dominant intentions and their relationships. This module applies the defined composition rules to generate the control flow of the initial composition schema.
- Module of semantic matching : This module performs the semantic matching between two concepts of an ontology according to matching levels (exact, subsumes, plugin, fail) and an ontology of the treated domain. It uses the Pellet reasoner to infer relationships between concepts. This module is used for the matching between intentions and abstract services and to calculate the semantic affinity between abstract services.

[1]Pellet: http://www.mindswap.org/2003/pellet/
[2]http://on.cs.unibas.ch/owls-api/

- Module of abstract services selection: this module ensures the selection of abstract services which will constitute the initial composition schema. To do this, it implements the step of matching between intentions and abstract services, and the step of selection based on affinity degree between abstract services. This module calls for the module of semantic matching.

- Module of initial composition schema refinement: it performs the refinement phase by selecting iteratively other abstract services of finer granularity. It calls for the semantic matching module to performing the matching. Furthermore, this module updates the data flow since of the composition schema.

- Module of concrete services selection : This module generates the execution plan by browsing the OWL file representing the final composition schema. It explores the web services registry.

6. Case study

We propose a case study concerning smart buildings in order to apply and partially validate the proposed approach. It especially deals with the optimization of the energy consumption in the building's rooms. Generally, each room is equipped with a set of lamps, a heater, air-conditioner and a window with blinds which are working automatically.

To apply our approach, we consider that the administrator of the building expresses his need to optimize the energy consumption in the building. A set of intentions is extracted (as shown in Table 5) and the correspondent graph of intentions G_0 (cf. Figure 14) is provided.

The intention I_1 dominates intentions I_3 (regulate luminosity) and I_5 (regulate temperature). To be able to regulate luminosity, the intention I_2 (calculate luminosity) precedes intention I_3. To be able to regulate temperature, the intention I_4 (calculate temperature) precedes I_5. I_5 dominates intention I_6 (Regulate Air-conditioning) and I_7 (Regulate heating system). This graph is provided with the information given in Table 5.

The first step of our approach consists on enriching the graph G_0 with the implicit relations between intentions using our enrichment module. This allows to add a precedence link between I_4 and I_6 and between I_4 and I_7 as shown in Figure 15. The second step begins by the construction of the control flow of the initial composition schema. The selection of abstract services is done using the semantic matching between intention goals and abstract service functionalities. For each intention, the semantic matching gives a set of abstract services. The selection is based on matching scores. It is based on a threshold. The selected set of abstract services corresponds to the services that have a matching score higher than the threshold. For the intention I_4, it is necessary to match the goal of the intention calculate temperature. The set of abstract services candidate are: s_{41} calculate temperature, s_{42} determine temperature, and s_{43} calculate heat degree. The matching scores are 2, 1.7, 1.5. The services

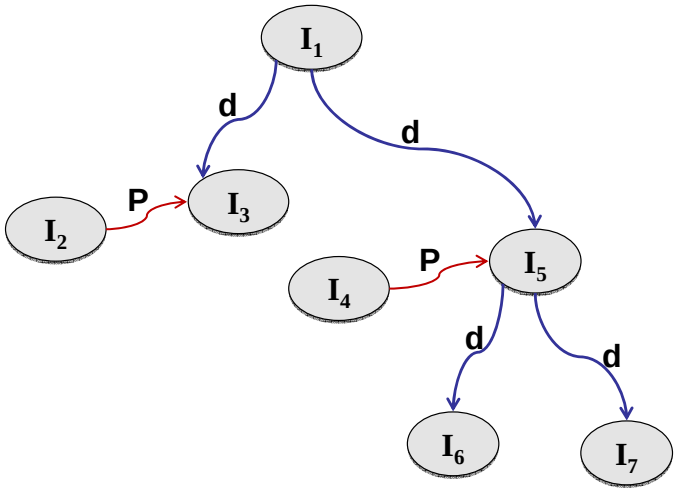

Figure 14: Original intentions graph

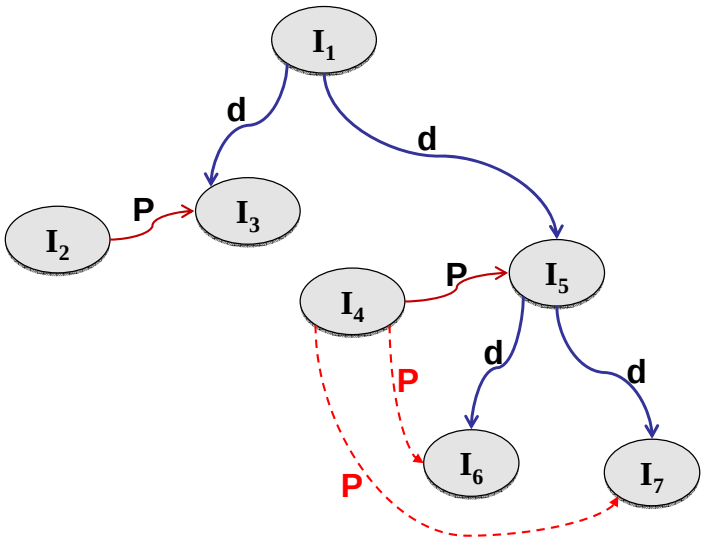

Figure 15: The case study intentions graph

s_{41} and s_{42} are selected because the threshold is 1.6. The operation of selecting abstract services is executed for each intention. At this stage, we have for each intention a set of services. For example we have the set $\{s_{41}, s_{42}\}$ precedes the set $\{s_{61}, s_{62}\}$ corresponding to I_6, and the set $\{s_{71}, s_{72}\}$ corresponding to I_7.

Table 5: The cas study intentions list

ID	Goal	Functional Constraints	Non Functional Constraints	Context parameters
I1	Optimize Consumption			
I2	Calculate luminosity	Precision=0.5 unit=lumen	Response Time<10	
I3	Regulate luminosity	LuminosityMax =1800 LuminosityMin =1200		Window blinds
I4	Calculate temperature	Precision=0.5 Unit=celsius		
I5	Regulate temperature	TempMax=25 TempMin=15		
I6	Regulate Air-Conditioning			Heater Air-Conditioner
I7	Regulate Heating System			Heater Air-Conditioner

For each combination generated with this example, we calculate the semantic affinity between outputs and inputs of connected services. Here, we explore 8 combinations, and the alternative selected presents the best affinity degree as we presented in section 5.2. In the end of this step we obtain the initial composition schema presented in Figure 16.

The next step is the generation of the final composition schema.

Our refinement module takes as input the initial composition schema (Figure 16). For example, the abstract service *Determine_luminosity* (s_{13}) is atomic. It will not be refined. The service *Regulate_Luminosity* (s_{22}) is a composite service with a *process* part Pr_{22} composed of three sub-processes: *Check_Presence* (SPr_{221}), *TurnOff_Lamp* (SPr_{222}) and *Adjust_luminosity* (SPr_{223}) organized as shown in Figure 17.

Our refinement module calls the semantic matching module for searching abstract services corresponding to SPr_{221}, SPr_{222}, and SPr_{223}. Several alternatives are found, and through the matching score and suitability to the context which consists of the existence of lamps (c.f. section 5.3), the best abstract services are selected. Services *Check_Presence* (s_{221}) and *TurnOff_Lamp* (s_{222}) are atomic, and the service *Adjust_luminosity* (s_{223}) is a composite service that will be refined in turn. The *process* part of the service s_{223} consists of two sub-processes: *Reduce_LampIntensity* (SPr_{2231}) and *Increase_luminosity* (SPr_{2232}) organized as shown in Figure 17.

The refinement module calls semantic matching module for processing SPr_{2231} and SPr_{2232}. The corresponding services found are s_{2231} which is

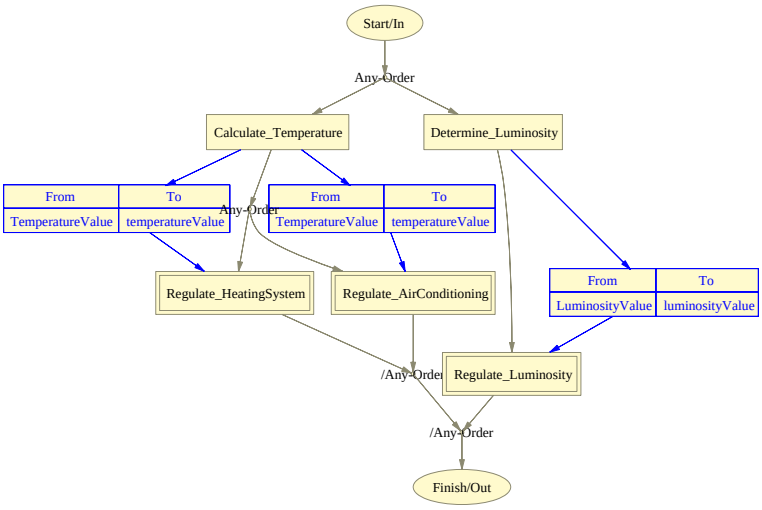

Figure 16: Initial composition schema

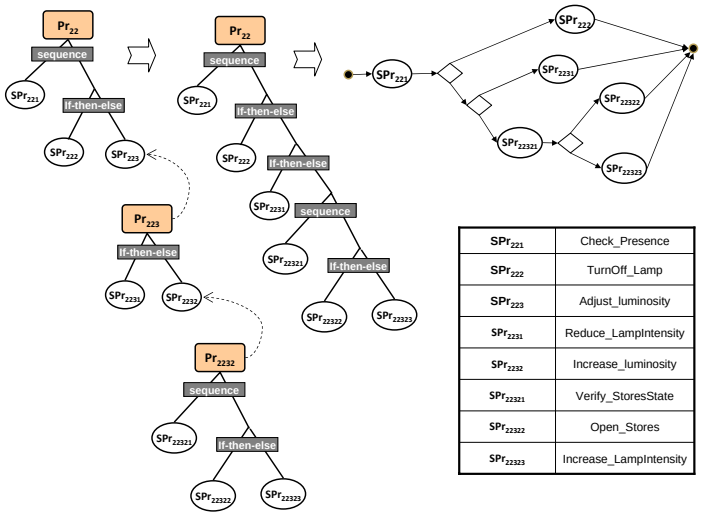

Figure 17: Refinement of the service *Regulate_Luminosity* (s_{22})

atomic, and s_{2232} which is composite. The *process* part of the service s_{2232} is composed of three sub-processes: *Verify_StoresState* (SPr_{22321}), *Open_Stores* (SPr_{22322}), and *Increase_LampIntensity* (SPr_{22323}) which are organized as

shown in Figure 17. The correspondence between processes and abstract service provides three atomic abstract services.

This completes the refinement of the abstract service *Regulate_Luminosity* (s_{22}). In fact, the service begins by verifying the state of the presence in the room, this determines if there is a need to run off the lamp or not. The adjustment of luminosity is made with respect to the constraint expressed by the intention I_3 which sets the maximum of luminosity degree. This service results in a reduction in the lamp intensity, or an increase of luminosity in the room (if the luminosity<LuminosityMin then increase, if > LuminosityMax then reducing). This action depends on the context parameter *BlindsState*, if the state is *closed*, then blinds will be opened, otherwise the lamp luminosity is increased.

The abstract service *Calculate_Temperature* (s_{42}) is atomic while services *Regulate_heatingSystem* (s_{61}) and *Regulate_AirConditioning* (s_{72}) are composite. The refinement module and the semantic matching module are used to refine these services until obtaining atomic services. In fact, the service s_{61} (resp. s_{72}) begins by verifying if the room is a lab, if it is the case, the heating is adjusted (resp. the cooling). This action depends on the existence of a heater (resp. air-conditioner) as we can see in I_6 (resp. I_7). The final composition schema is given in Figure 18.

7. Experimental results

We have defined an architecture that implements the features presented in previous sections. The different modules included in the architecture have been implemented using Java language and specific API's such as OWL API[3] and OWL-S API[4].

We use the Smart Building case study to generate sets of intentions graphs from 7 nodes graph sets to 20 nodes graph sets. We consider that an intentions graph of 20 nodes is sufficient for a Smart Building. We execute our approach on each set and we obtain results showed in Figure 19. For smaller graphs (under 11 nodes), we remark that execution time does not exceed 4 seconds. Graphs with a size between 12 and 16 have an average execution time around 5 seconds. Graphs with a size between 17 and 20 have an average execution time around 9 seconds. The duration of enrichment and the duration of final composition schema generation are more important than the duration of the initial composition schema generation. For the enrichment, this is due to the exploration of the graph. For the final composition schema generation, the fact that this task requires consecutive substitutions of abstract services subprocesses based on semantic matching increase the time of execution. In the initial composition schema generation, only two operations of semantic matching are required that reflect on the execution time.

[3]http://owlapi.sourceforge.net/

[4]http://www.mindswap.org/2004/owl-s/api/

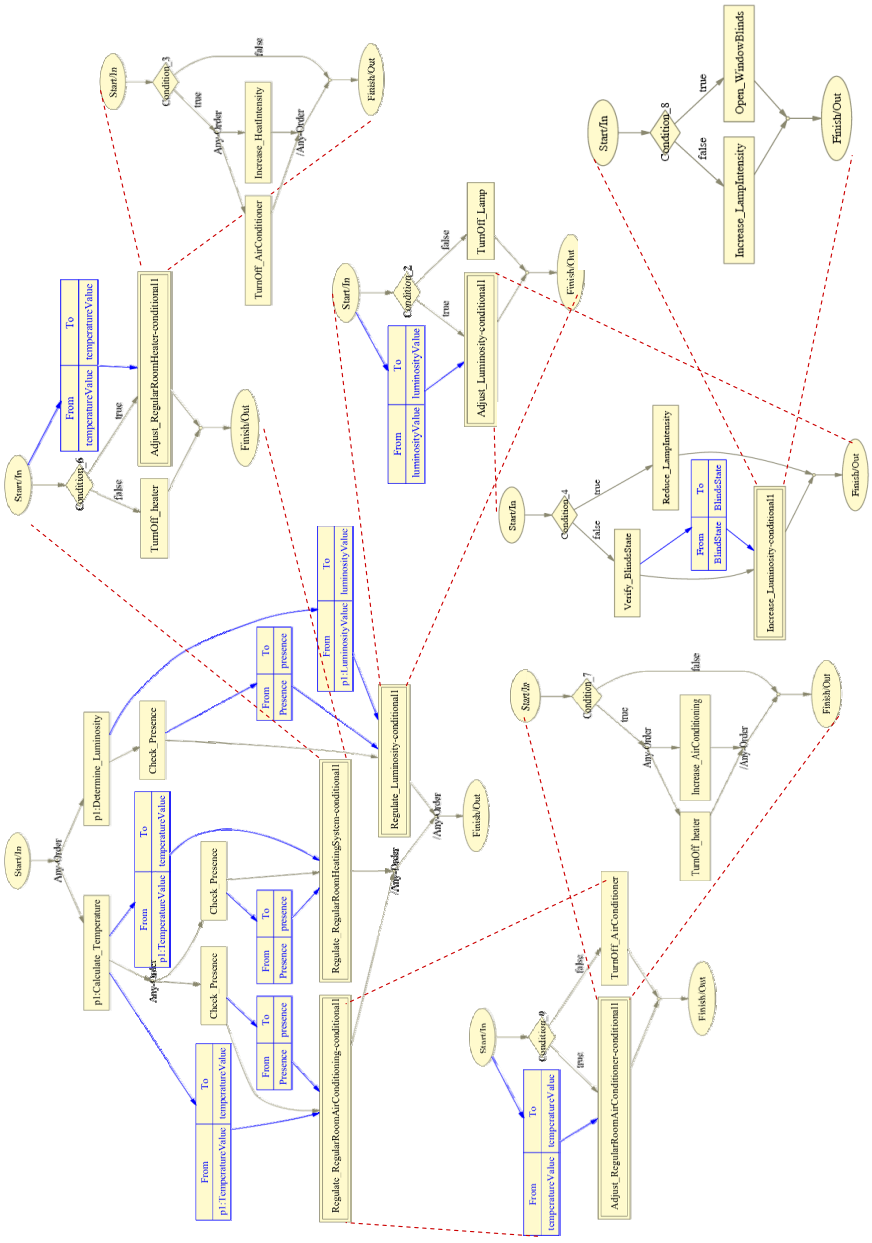

Figure 18: The final composition schema

In order to assess the quality of results, we studied the metrics *precision* and *recall* [11] to see the quality of the generation of the initial composition schema.

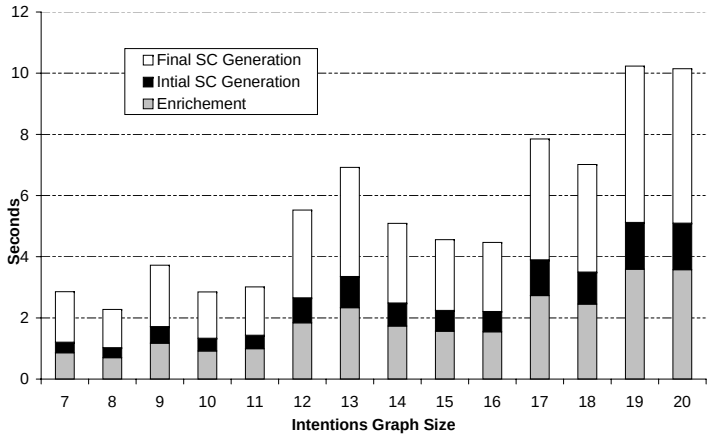

Figure 19: Execution time of composition process

These two metrics are defined by two sets: the set of found composition schemas and the set of relevant composition schemas. The metric *precision* indicates the ability of the composition process to find only relevant composition schemas without considering the false positives. The metric *recall* measures the ability of the composition process to find all relevant composition schemas.

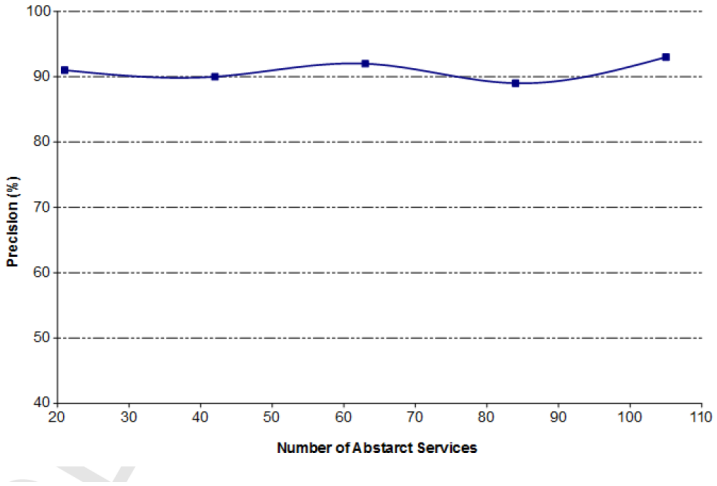

Figure 20: Measuring the metric *precision*

The metric *precision* is defined as follows:

$$\frac{\{\text{Relevant composition schemas}\} \cap \{\text{Found composition schemas}\}}{\{\text{Found composition schemas}\}}$$

The metric *recall* is defined as follows:

$$\frac{\{\text{Relevant composition schemas}\} \cap \{\text{Found composition schemas}\}}{\{\text{Relevant composition schemas}\}}$$

To assess these metrics, we have defined for each intention of case study the graph 15 abstract services as a maximum. So we have 7 intentions $I_1..I_7$ with 15 abstract services as a maximum for each intention.

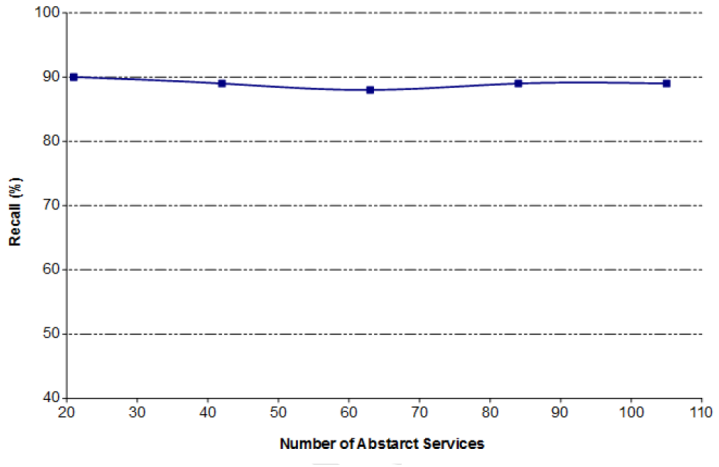

Figure 21: Measuring the metric *recall*

We conducted experiments with a matching threshold equal to 1.5.

In the scenario of the case study, the building administrator wants to reduce energy consumption by using only the adequate devices. The results shown in Figure 20 indicate that our approach for the generation of the initial composition schema shows a level of *precision* with an average value of 91%. This result indicates that our approach of the initial composition schema has a great chance to obtain the most appropriate composition schema for the user. This is due to the choice of the matching threshold that is fairly high. The good result for the *precision* is confirmed by the result on the *recall* that its average is close to 89% , as shown in Figure 21.

Combined analysis of metrics *precision* and *recall* shows that the composition mechanism has managed to generate a significant number of compositions that meet user needs with a low rate of false positives. We believe that the proposed composition mechanism allows to generate the compositions corresponding to to the needs of the user, and this through the use of semantic matching between intention and abstract service and through the use of semantic affinity between services.

8. Related Work

Automatic composition of services is a popular research topic that receives a lot of attention and that has been studied for several years. Most of current service composition approaches require predefined process schemas to construct composite Web services. Authors in [12] propose an approach for service composition based on the selection the best execution plan. This selection is QoS aware. The problem of QoS-aware service composition is modeled as a constraint satisfaction framework. However, the only aspect considered is QoS, and the method requires a predefined workflow. Another aspect is further considered in [13] for dynamic service composition, which is transaction aspect. In this work, authors formulate the problem of transactional and QoS-aware dynamic service composition as a constrained directed acyclic graph. Although the obtained results are interesting, the approach is based on a predefined composition schema.

A quality of service (QoS)-aware execution plan selection approach for a service composition process

This subject involves in and profits from many techniques and topics of computer science such as Artificial Intelligence, semantic web and ontology [14]. The major categories of approaches for automatic service composition are: artificial intelligence (AI) planning techniques, chaining based techniques and knowledge based approaches in general.

Several research studies have exploited the AI planning techniques to solve the problem of the composition of services like [15], [16] and [17]. An AI-problem planning is defined by an initial state, a target state representing the goal of the plan and a set of actions. The objective is to find a path from the initial state to the target state. This path is the action plan which represents a sequence of actions. A composite Web service, in general, is similar to a state-transition system. It presents different states and actions in certain states and represents transitions from an initial state to a final state to providing required outputs. Thus, many studies on automated service composition have focused on solving the problem of the composition by converting it into a problem of finding appropriate transition systems. Hence, different AI techniques have been proposed such as contingency planning, HTN planning (Hyper Task Network), proof of linear logic theorem , and constraint based programming [16, 18, 19, 20, 21, 22, 23]. [16], [21], [24] and [17] propose an approach for Web service composition which takes into account the semantic description of service functionality based on HTN planning. HTN is a planning method which predefines the decomposition of every service. It is not always possible to identify the functionality of every service and the decomposition relationships between all services to be able to construct a HTN. The general service composition is supposed to provide a new functionality by composing different services from a new user's request. Therefore, the above mentioned approach is not suitable for the general service composition. Our approach has similarity with HTN method in that the composition or the decomposition of each composite abstract service is predefined. In the other hand, in our approach, we have the following

advantages: (1) each abstract service presents a fragment of process (of a certain granularity level) which can be used by many composite services, (2) A user request corresponds to many abstract services which must be combined at processing time. Indeed, in our approach, the composition schema generation is achieved partly at run-time while having abstract services as design-time components. This offers a certain flexibility and adaptability in composition without having to deal with composition from scratch at run-time. Rao et al. [18] convert service specifications into axioms and user needs into linear logic theorems and try to find an adequate composite service using theorem proving. Akkiraju et al. [20] tries to improve the semantic precision of the resulting composite services using semantic matching in the planning process. Song et al. [25] present a workflow framework for service composition. This framework is composed of a planning module and CSP (Constraint Satisfaction Problems) solving module. However the planning is performed at design time without considering contextual information. In [22], constraint logic programming is used in order to find appropriate services and construct composite services. The establishment of a composite service is based on checking whether required outputs are reachable from the inputs in hand using services. Then they build an appropriate composite service based on the reachability. The major limitation of the above approaches is the assumption that each service has pre-conditions and effects. If available services cannot have pre-conditions and effects, only input and output parameters matching are used for composition. The generated composition may not be satisfying to the user intention since there is services having same input and output parameters while having different functionality. In our work, we explicitly define functional semantics of our abstract services to allow selecting those services that meet user intentions. Moreover, a user does not actually express his intentions in terms of I/O. Thus, in our opinion, more emphasis should be put on the importance of the functional aspect of services for composition. Another limitation of using AI planning in automatic service composition is that AI planning, in general, generates a sequence of atomic actions and does not consider contextual information. Moreover, plans may require complex structures of control like choice, non-determinism and loops. In our approach, we tackle this problem by encapsulating complex control structures in abstract services (at design-time) and chaining services at run-time.

Many research studies have applied techniques based on chaining in service composition [26], [27], [28], [29]. They try to find dependencies between different services in order to build a composition plan. [26] proposes the search of a composition plan applying the shortest path algorithm on a graph of services. The exploration of the graph is based on a forward-chaining algorithm. In [29], backward chaining is used to explore all possible compositions and available services are determined during the search process. However, the response time is too long since a great number of independent services are available. In addition, these approaches can not guarantee that generated composite services provide correctly the requested functionality, since they consider the matching and dependencies between input and output parameters regardless of functional semantics of each service.

9. Conclusion

In this paper, we have proposed a solution for automated and adaptable service composition. The composition mechanism relies on abstract services that feature semantic, generic and reusable descriptions. This allows, on one hand, the specification of generic processes for different situations, and on the other hand, selecting adequate services that meet user intentions. We claim that automated composition mechanisms have to be provided to adapt to user requirements and situations. Our composition process is guided by a specification of user intentions and ignores the problems related to extraction of these intentions from requests or contexts. We focus in this paper on the steps to perform the generation of the composition schema. This generation is based mainly on semantic matching mechanisms.

Future research work will focus furthermore on the study of an efficient selection policy for the concrete Web services that takes into account the volatility aspect of concrete Web services. Indeed, this selection policy should minimize the impact of adding and removing of Web services, by service providers, on the definition of abstract services.

References

[1] A. Bucchiarone, S. Gnesi, A Survey on Services Composition Languages and Models, in: Proceedings of International Workshop on Web Services Modeling and Testing 2006 (WS-MaTe 2006), 2006.

[2] I. J. G. dos Santos, M. Flügge, N. P. Tizzo, E. R. M. Madeira, Challenges and techniques on the road to dynamically compose web services, in: Proceedings of the 6th International Conference on Web Engineering, ICWE 2006, Palo Alto, California, USA, ACM, 2006, pp. 40–47.

[3] S. R. Ponnekanti, A. Fox, Sword: A developer toolkit for web service composition, in: Proceedings of the 11th International WWW Conference (WWW2002), Honolulu, HI, USA, 2002.

[4] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al., OWL-S: Semantic Markup for Web Services.

[5] L. Ye, B. Z. 0001, Discovering web services based on functional semantics., in: APSCC, IEEE, 2006, pp. 348–355.

[6] B. Grosz, C. Sidner, Attention, intentions, and the structure of discourse, Computational Linguistics 12 (3).

[7] H. Kanso, C. Soul-Dupuy, S. Tazi, Reconnaissance des intentions de communication dans des corpus de documents scientifiques, in: I. Saleh, K. Ghedira, B. Badreddine, N. Bouhai (Eds.), Collaborer, Echanger, Inventer (H2PTM), Hammamet (Tunisie), 29/10/07-31/10/07, Herms Science Publications, http://www.editions-hermes.fr/, 2007, pp. 387–398.

[8] K. M. Sim, P. T. Wong, Web-based information retrieval using agent and ontology., in: N. Zhong, Y. Yao, J. Liu, S. Ohsuga (Eds.), Web Intelligence, Vol. 2198 of Lecture Notes in Computer Science, Springer, 2001, pp. 384–388.

[9] M. Paolucci, T. Kawamura, T. R. Payne, K. P. Sycara, Semantic matching of web services capabilities, in: International Semantic Web Conference, 2002, pp. 333–347.

[10] R. Rada, H. Mili, E. Bicknell, M. Blettner, Development and application of a metric on semantic nets, in: IEEE Transactions on Systems, Man and Cybernetics, 1989, pp. 17–30.

[11] F. Salfner, M. Lenk, M. Malek, A survey of online failure prediction methods, ACM Comput. Surv. 42 (3) (2010) 10:1–10:42.

[12] M. Liu, M. Wang, W. Shen, N. Luo, J. Yan, A quality of service qos-aware execution plan selection approach for a service composition process, Future Gener. Comput. Syst. 28 (7) (2012) 1080–1089.

[13] Q. Wu, Q. Zhu, Transactional and qos-aware dynamic service composition based on ant colony optimization, Future Gener. Comput. Syst. 29 (5) (2013) 1112–1119.

[14] D. Zhovtobryukh, A petri net-based approach for automated goal-driven web service composition, Simulation 83 (1) (2007) 33–63. doi:10.1177/0037549707079226.
URL http://dx.doi.org/10.1177/0037549707079226

[15] J. Peer, A pddl based tool for automatic web service composition, in: In Proceedings of the Second Intl Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR, Springer Verlag, 2004, pp. 149–163.

[16] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau, Htn planning for web service composition using shop2, Web Semantics: Science, Services and Agents on the World Wide Web 1 (4) (2004) 377–396. doi:10.1016/j.websem.2004.06.005.
URL http://dx.doi.org/10.1016/j.websem.2004.06.005

[17] X. Tang, F. Tang, L. Bing, D. Chen, Dynamic web service composition based on service integration and htn planning, in: 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013, pp. 307–312. doi:10.1109/IMIS.2013.58.

[18] J. Rao, P. Kngas, Logic-based web services composition: From service description to process model, in: In Intl. Conference on Web Services (ICWS, IEEE, 2004, pp. 446–453.

[19] L. A. G. Da Costa, P. F. Pires, M. Mattoso, Automatic composition of web services with contingency plans., in: ICWS, IEEE Computer Society, 2004, pp. 454–461.

[20] R. Akkiraju, A. Ivan, R. Goodwin, B. Srivastava, T. F. Syeda-Mahmood, Semantic matching to achieve web service discovery and composition., in: CEC/EEE, IEEE Computer Society, 2006, p. 70.

[21] R. Thiagarajan, M. Stumptner, Service composition with consistency-based matchmaking: A csp-based approach, in: Web Services, 2007. ECOWS '07. Fifth European Conference on, 2007, pp. 23 –32. doi:10.1109/ECOWS.2007.26.

[22] S. Kona, A. Bansal, G. Gupta, Automatic composition of semantic web services, in: Web Services, 2007. ICWS 2007. IEEE International Conference on, 2007, pp. 150–158. doi:10.1109/ICWS.2007.52.

[23] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, B. Srivastava, A service creation environment based on end to end composition of web services, in: Proceedings of the 14th International Conference on World Wide Web, WWW '05, ACM, New York, NY, USA, 2005, pp. 128–137. doi:10.1145/1060745.1060768.
URL http://doi.acm.org/10.1145/1060745.1060768

[24] Y. Xiao, X. Zhou, X. Huang, Automated semantic web service composition based on enhanced htn, in: 2010 Fifth IEEE International Symposium on Service Oriented System Engineering, 2010, pp. 59–63. doi:10.1109/SOSE.2010.48.

[25] X. Song, W. Dou, J. Chen, A workflow framework for intelligent service composition, Future Generation Computer Systems 27 (5) (2011) 627 – 636. doi:https://doi.org/10.1016/j.future.2010.06.008.
URL http://www.sciencedirect.com/science/article/pii/S0167739X10001214

[26] I. B. Arpinar, R. Zhang, B. Aleman-Meza, A. Maduko, Ontology-driven web services composition platform, Inf. Syst. E-Business Management 3 (2) (2005) 175–199.
URL http://dblp.uni-trier.de/db/journals/isem/isem3.html#ArpinarZAM05

[27] L. Aversano, G. Canfora, A. Ciampi, An algorithm for web service discovery through their composition, 2013 IEEE 20th International Conference on Web Services 0 (2004) 332. doi:http://doi.ieeecomputersociety.org/10.1109/ICWS.2004.1314755.

[28] S. V. Hashemian, F. Mavaddat, A graph-based framework for composition of stateless web services, in: Proceedings of the European Conference on Web Services, ECOWS '06, IEEE Computer Society, Washington, DC,

USA, 2006, pp. 75–86. doi:10.1109/ECOWS.2006.2.
URL https://doi.org/10.1109/ECOWS.2006.2

[29] F. Mohr, A. Jungmann, H. K. Bning, Automated online service composition, in: 2015 IEEE International Conference on Services Computing, 2015, pp. 57–64. doi:10.1109/SCC.2015.18.