A Framework for Dynamic Web Services Composition

Freddy Lécué[1], Eduardo Silva[2], and Luís Ferreira Pires[2]

[1] France Telecom R&D, France
Rue du clos courtel, F-35512 Cesson Sévigné
freddy.lecue@orange-ftgroup.com
[2] Centre for Telematics and Information Technology
University of Twente, The Netherlands
P.O. Box 217, 7500 AE Enschede
{e.m.g.silva, l.ferreirapires}@ewi.utwente.nl

Abstract. Dynamic composition of web services is a promising approach and at the same time a challenging research area for the dissemination of service-oriented applications. It is widely recognised that service semantics is a key element for the dynamic composition of Web services, since it allows the unambiguous descriptions of a service's capabilities and parameters. This paper introduces a framework for performing dynamic service composition by exploiting the semantic matchmaking between service parameters (i.e., outputs and inputs) to enable their interconnection and interaction. The basic assumption of the framework is that matchmaking enables finding semantic compatibilities among independently defined service descriptions. We also developed a composition algorithm that follows a semantic graph-based approach, in which a graph represents service compositions and the nodes of this graph represent semantic connections between services. Moreover, functional and non-functional properties of services are considered, to enable the computation of relevant and most suitable service compositions for some service request. The suggested end-to-end functional level service composition framework is illustrated with a realistic application scenario from the IST SPICE project.

1 Introduction

An important benefit of the Service-Oriented Architecture (SOA) is that it enables dynamic service binding, which allows service users to discover, select and invoke services at runtime. Web services technologies [1] provide a suitable technical foundation for developing and deploying loosely coupled and reusable software components, which can be invoked through their service ports. *Web services* are distributed and programmatically accessible over standard Internet protocols, and interoperate independently of the programming languages, operating systems and hardware platforms used to implement them. Therefore, Web services technologies offer the feature richness, flexibility and scalability needed by enterprises to profit from the SOA benefits.

Automated service discovery, selection and composition are expected to enrich the experience of service end-users through value-added services, and to allow automated processes to interact with minimal human intervention [2]. However, some work still

has to be done to appropriately support dynamic and automated service discovery, selection and composition with the current Web services technologies. The automation of these tasks requires some knowledge about the services, such as: (i) description of the service capabilities, for example, in terms of the semantics of IOPEs (Input, Output, Preconditions and Effects); (ii) process model, which provides a description of the service activities, interaction protocol and exchanged messages; (iii) grounding specification of the service, which describes the coding used to map information onto messages and the protocols used to exchange these messages. These requirements are expected to be covered by defining semantic models of web services, using techniques from the *Semantic Web services* [3]. A Semantic web service is a web service described in a language with well-defined semantics. This feature of the Semantic web services enables different kinds of inference and reasoning based on the service semantic descriptions, in order to facilitate dynamic service discovery, selection and composition.

In order to tackle the challenge of service composition, most of the work done until now has focused on two main composition approaches, namely by considering functional [4–8] and process [9–12] service aspects. The approach based on functional aspects aims at finding a sequence of atomic components described in terms of their IOPEs that matches a given query. This sequence can be executed from the start conditions provided by the query, so that the query goal is satisfied at the end of the sequence. The approach based on process aspects considers services as stateful processes with a choreography represented in terms of sequential, conditional, and iterative steps imposed by the service. These two composition approaches are complementary and form an interesting trade-off to develop solutions for service composition [13].

In this paper we focus on a framework for service composition based on functional aspects, in which services are chained according to their functional description (IOPEs). The suggested framework uses the Causal Link Matrix (CLM) formalism [14] in order to facilitate the computation of the final service composition as a semantic graph. The nodes of this semantic graph represent semantic connections between component services. By computing a CLM we increase the amount of relevant service compositions that can be obtained. The set of possible solutions are pruned, at composition time, in order to rank the service compositions according to an optimization criteria. These criteria can be defined based on the semantic similarity of component services and/or the non-functional properties of the compositions calculated by aggregating the non-functional properties of the component services.

The rest of the paper is organized as follows: Section 2 motivates our framework with application scenarios and an example; Section 3 introduces the SPICE Automatic Composition Engine (ACE) architecture in which our framework is used; Section 4 presents our framework for dynamic service composition; Section 5 comments on related work, and; Section 6 gives some final remarks.

2 Motivation

Dynamic composition of services aims at composing services that satisfy a given service request from an end-user or service developer. Services are composed of existing atomic services, which are orchestrated in the service composition.

Once dynamic service composition mechanisms are available, the service creation task performed by end-users and service developers is expected to be simplified. In this paper we specially focus on the service developer scenario that we are developing on the IST SPICE [15] project. In this scenario a service developer aims at creating a new service with some specific functional and non-functional properties. To achieve this, a formalism should be used to describe these properties in a service request, specifying these properties unambiguously to allow automatic reasoning based on the service request. After the service request is specified, the framework for dynamic service composition is capable of discovering, matching and composing a set of services that together fulfil the request. The resulting compositions are returned to the service developer, who should select the composition that best fits his needs. The service developer may adapt the selected composition further to fulfil more specific requirements. This process provides a service developer with a tool for automatically finding and composing a set of services that meet his needs, relieving him from the burden of manually dealing with the whole service creation cycle.

2.1 Example

We consider an example in which a service developer wants to develop a new service that receives a piece of text, translates it to English, and sends the translated text by SMS to a given destination number. In case no support is available for the service composition, the service developer is forced to create the service by scratch by connecting the available atomic services in the service composition implementation manually. In case an orchestration language such as WS-BPEL [16] is available, the service developer can specify the service composition in terms of an orchestration of the atomic services. In our example this corresponds to an orchestration of the translation services and SMS messaging services. The objective of our framework is to go one step further and automatically generate service compositions that cope with the service developer service request and also meet the non-functional properties (e.g., cost, response time, etc.) specified in the service request.

2.2 Service Request

Service developers specify service requests in terms of annotations that define the requested service inputs, outputs, goals, preconditions, effects and ontologies. These annotations are references to elements defined on ontologies described in OWL [17]. An example of annotated service request is:

```
<Input>
   <"LanguageOnt#Language" name="srcLang">
   <"LanguageOnt#English" name="trgtLang">
   <"LanguageOnt#Text" name="txtToTrans">
   <"TelecomOnt#PhoneNum" name="destNumber">
</Input>
<Output>
   <"TelecomOnt#AckSMS" name="AcknowledgmentSMS">
</Output>
<Preconditions/><Effects/>
<Goal>
   <"GoalOnt#translate">
```

```
   <"GoalOnt#sendSMS">
</Goal>
<Non-functional>
   <"NFPOnt#Cost" value=6>
</Non-functional>
<Ontologies>
   <"GoalOnt" "TelecomOnt" "NFPOnt" "LanguageOnt">
</Ontologies>
```

These annotations indicate that the service developer requests a service that translates a piece of text to English and sends the translated text by SMS to a given destination number. This is the running example used to illustrate our framework for dynamic service composition in this paper.

3 Automatic Service Composition Engine

The aim of SPICE is to provide a platform to support the development and deployment of innovative and value-added services during their whole life cycle. The creation and development of services is achieved in a service creation environment, which allows the manual creation of services for end-users and service developers. The service creation environment also contains an Automatic Composition Engine (ACE), which automatically constructs a service that fits a service request issued by end-users or service developers.

The SPICE ACE contains four basic components: *Semantic Analyser*, *Composition Factory*, *Property Aggregator* and *Matcher*. Figure 1 depicts the ACE architecture.

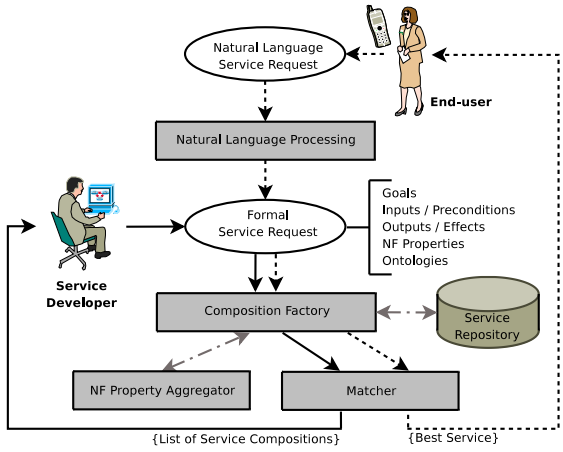

Fig. 1. SPICE ACE Architecture

Figure 1 shows the two basic ACE usage scenarios: (i) an end-user issues a service request in natural language (at runtime) and gets the most suitable service composition,

or (ii) a service developer issues a service request in some well-defined formalism (at design-time) and gets a set of relevant service compositions.

The end-user is shielded from the complexity of the composition process by being allowed to request services in natural language. These requests are processed by the Semantic Analyser, which constructs a formal service request according to the ACE's service request formalism. The resulting formal request follows the same structure used by the service developer for defining service requests.

When a formal service request is defined, the Composition Factory queries the service repository for a service that matches the service request. If a match exists on the repository, the matching service is returned. In case no match is found, the Composition Factory creates a composite service that matches the request. In principle, the Composition Factory may generate multiple alternative compositions that match a service request.

Services and service requests are characterized by their functional and non-functional properties. Functional properties are the services' goals, inputs, outputs, preconditions and effects. These properties are used to perform the service discovery, matching and composition. Examples of non-functional properties are cost, security, performance, reliability, etc. Non-functional properties are used to limit the space of compositions that fulfil the service request, and to rank the generated set of compositions. Service and service request descriptions also contain the domain ontologies used to define the functional and non-functional properties in an unambiguous form.

The Composition Factory uses the Property Aggregator to compute the non-functional properties of service compositions each time a new service is added to a service composition. The non-functional properties of the resulting service composition are calculated by aggregating the non-functional properties of the atomic component services.

The set of generated service compositions is then passed to the Matcher component, which matches each service composition with the service request, using the aggregated non-functional properties and the measures of semantic similarity. In the scenario where the end-user requests a service, the best matching is returned to the end-user. This matching is obtained by taking the user's profile and context information into consideration, which are managed by the SPICE platform. In the scenario where the service developer issues a service request, the full set of generated compositions is returned, possibly ranked taking into account the resulting aggregated non-functional properties and/or the measures of semantic similarity.

4 Dynamic Web Service Composition

The Composition Factory component is responsible for the creation of service compositions based on a formal service request, and is the focus of this section. After receiving the developer's service request, the Composition Factory queries the service repository in order to retrieve an unordered set of services required to compute the service composition. Semantic connections between web services are stored on a CLM^{+}, which is then used to compute the semantic graph-based composition that represents the possible service compositions matching the service request. Figure 2 gives an overview of the steps performed by our dynamic service composition framework.

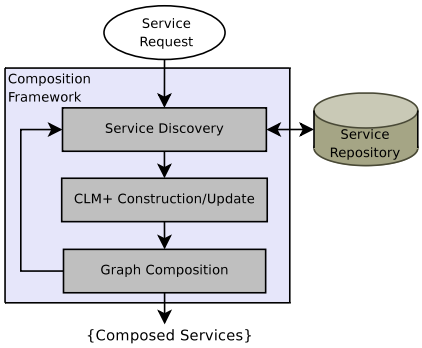

Fig. 2. Service Composition Framework

4.1 Causal Links

When using functional composition approaches, semantic connections between different component web services are the main issue to be handled in order to create new value-added web services. These connections are mainly useful to semantically link output to input parameters of web services, creating in this way simple sequential compositions of web services. A composition is defined as an ordered set of web services in which the web services of this set are semantically linked to each other.

Input and Output parameter types of semantic web services are concepts defined in an ontology $\mathcal{T}$. These parameter types can be represented by using some standard language, such as, e.g., OWL-S [18] (at profile level), WSML [19] (at capability level), or SA-WSDL [20]. Retrieving the semantic connection between two Web services s_x and s_y is similar to discovering the semantic similarity between an output parameter Out_s_y of s_y and an input parameter In_s_x of s_x (or vice-versa). Consequently, our goal is to find a matchmaking [21] function between two knowledge representations encoded using the same ontology $\mathcal{T}$. Causal links[3] [14] between web services not only value these semantic matchmaking functions, but also measure the quality of semantic links between web services. In other words, a causal link (see figure 3) describes a semantic relation between an output parameter $Out\_s_y \in \mathcal{T}$ of a service s_y and an input parameter $In\_s_x \in \mathcal{T}$ of a service s_x. Thereby s_x and s_y are semantically and partially linked according to a matchmaking function $Sim_{\mathcal{T}}(Out\_s_y, In\_s_x)$. The matchmaking function $Sim_{\mathcal{T}}$ determines the matchmaking type [23, 24] between these two parameters, and can have the following values:

- **Exact ($\equiv$)** if the output parameter Out_s_y of s_y and the input parameter In_s_x of s_x are equivalent concepts; formally, $\mathcal{T} \models Out\_s_y \equiv In\_s_x$.
- **PlugIn ($\sqsubseteq$)** if Out_s_y is sub-concept of In_s_x; formally, $\mathcal{T} \models Out\_s_y \sqsubseteq In\_s_x$.
- **Subsume ($\sqsupseteq$)** if Out_s_y is super-concept of In_s_x; formally, $\mathcal{T} \models In\_s_x \sqsubseteq Out\_s_y$.
- **Intersection ($\sqcap$)** if the intersection of Out_s_y and In_s_x is satisfiable; formally, $\mathcal{T} \not\models Out\_s_y \sqcap In\_s_x \sqsubseteq \bot$.

[3] In AI planning area, causal links are sometimes called *protection intervals* [22].

– **Disjoint ($\perp$)** if Out_s_y and In_s_x are incompatible; formally, $\mathcal{T} \models Out\_s_y \sqcap In\_s_x \sqsubseteq \perp$.

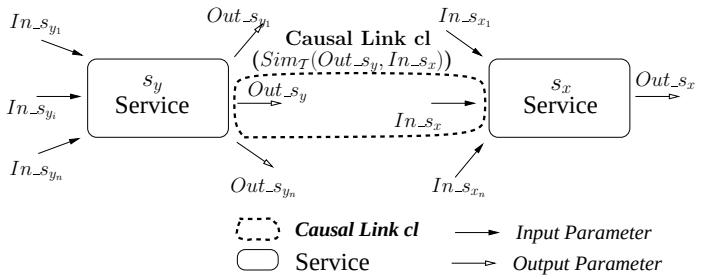

Fig. 3. Causal Link.

Since a causal link is related to a logical dependency among input and output parameters of different web services, [14] defines a causal link as a triple $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$. s_x and s_y refer to two web services in a set of available web services S_{Ws}. The concept Out_s_y is an output parameter of the service s_y whereas the concept In_s_x is an input parameter of the service s_x. The matchmaking function $Sim_{\mathcal{T}}$ returns the matching type depending on the matching degree between the concepts $Out\_s_y, In\_s_x \in \mathcal{T}$. A causal link $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$ implies that (a) s_y precedes s_x, since an output of s_y is consumed by an input of s_x, and (b) no web service call is planned between s_x and s_y.

Definition 1 *(Valid Causal link)*
A causal link $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$ is valid iff $Sim_{\mathcal{T}}(Out\_s_y, In\_s_x)$ is not a $Disjoint$ matchmaking.

The matchmaking type returned by the causal link is useful to value the possible semantic connection between two web services and also to compare links. Considering two web services s_y and s_z with their respective output parameters Out_s_y and Out_s_z. Considering a service s_x so that both Out_s_y and Out_s_z semantically match with In_s_x, $Sim_{\mathcal{T}}$ is able to quantify the two connections (Out_s_y, In_s_x) and (Out_s_z, In_s_x) and also to order them with respect to the matchmaking.

Although the matchmakings $Exact$, $PlugIn$, and $Disjoint$ can be used without any change to value causal links in a web service composition, causal links valued as Intersection or Subsume (also known as non-robust causal links) need some refinements to be fully efficient for causal links composition. Further details on web service composition with non-robust causal links are given in [25].

Since a composition of web services consists of a partial order of web services in which these services are semantically chained by causal links, web service composition can be considered as a composition of causal links. Therefore in this paper we simply reuse and extend the CLM model to store the causal links that are relevant for service composition.

4.2 Service Discovery

We perform service discovery based on the service request goals in order to discover candidate services for the composition. To discover these services we assume that all the services in the service repository have a semantic goal description and references to ontologies, which can be used to search and discover the relevant services. Our framework does not support service discovery; we simply assume the availability of functionality to perform goal-based discovery in the service execution environment. In SPICE, ontology-based discovery is expected to be supported by a discovery facility.

In our running example, two main goals have been defined for the service request: $GoalOnt\#translate$ and $GoalOnt\#sendSMS$. Using these semantic annotations, the repository is queried for existing services that cope with these goals, or services that have goals semantically close to these goals. We assume that a set of services S_{Ws} is returned, and no single service fully matches the service request. Table 1 shows a possible list of discovered services S_{Ws}, with respective inputs, outputs and non-functional properties semantic types and values.

Table 1. Discovered Services

Service	Input	Output	NF properties
S_1	LanguageOnt#Language LanguageOnt#English LanguageOnt#Text	LanguageOnt#EnglishText	NFPOnt#Cost 1
S_2	LanguageOnt#French LanguageOnt#English LanguageOnt#Text	LanguageOnt#EnglishText	NFPOnt#Cost 4
S_3	TelecomOnt#PhoneNum LanguageOnt#Text	TelecomOnt#AckSMS	NFPOnt#Cost 1
S_4	TelecomOnt#PhoneNum LanguageOnt#Text	TelecomOnt#AckSMS	NFPOnt#Cost 3
S_5	TelecomOnt#AckSMS	TelecomOnt#SuccessProcess	NFPOnt#Cost 1

Table 1 shows that service S_1 is responsible for translating any text in any language to English, whereas S_2 translates text from French to English. These web services refer to three simple $\mathcal{FL}_0$ ontologies, namely $LanguageOnt$, $TelecomOnt$ and $NFPOnt$. The properties of these parameters are: $EnglishText \sqsubset Text$, $French \sqsubset Language$ and $Cost \sqsubset NFProperty$.

4.3 CLM and Non-functional Parameters

We extend the definition of CLM [14] below by considering not only causal links but also non-functional parameters of services. In this way, a CLM extended with non-functional parameters, denoted as CLM$^+$ (definition 2), can be used in the automated web service composition process by classifying web services in an appropriate way, according to the causal link and the services' non-functional parameters. All causal links are pre-computed in the CLM$^+$ to facilitate web service composition. The more valid causal links can be found, the better the solution to the functional composition problem.

Table 2. Labels of the rows r_i and columns c_j of the 6×6 matrix $\mathcal{M}$.

i/j index	1	2	3	4	5	6
$r_{i.label}/c_{i.label}$	Language	French	English	Text	PhoneNum	AckSMS

Definition 2 ***(CLM$^+$)***
An extended CLM (CLM$^+$), $\mathrm{M}_{p,p}$*, is defined as a* $p \times p$ *matrix of elements* $m_{i,j}$*, which are a set of triplets* $(s_y, score, \boldsymbol{q}_{s_y}) \in S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re^n$ *with*

$$(s_y, score, \boldsymbol{q}_{s_y}) = (s_y, Sim_{\mathcal{T}}(Out\_s_y, c_j), \boldsymbol{q}_{s_y})$$

Columns $c_{j,j\in\{1,...,p\}}$ *and rows* $r_{i,i\in\{1,...,p\}}$ *are both labelled by* $Input(S_{Ws}) \subseteq \mathcal{T}$ *i.e., the inputs parameters of services* S_{Ws}*;* $r_i \in \mathcal{T} \cap In(s_y)$ *is the label of the* i^{th} *row such that* $In(s_y)$ *is the set of input parameters of* s_y*; and* $c_j \in \mathcal{T} \cap$ *(Input(*S_{Ws}*)) is the label of the* j^{th} *column,* $Out\_s_y \in Out(s_y)$*.*

A CLM$^+$ is a matrix with entries in $\mathcal{P}(S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re^n)$. Each entry of the matrix refers to a set of triples $(s_y, score, \boldsymbol{q}_{s_y})$, such that the score represents the semantic similarity between an output parameter $Out\_s_y \in Out(s_y)$ of a web service s_y and an input parameter of another web service in S_{Ws}. Therefore a CLM$^+$ pre-computes the semantic similarities between all output and input parameters of a closed set of web services, i.e., a set of relevant web services for composition. According to definition 2, a CLM$^+$ contains all enabled, legal and valid links since causal links with a $Disjoint$ score are omitted in the CLM$^+$. The value of causal links $Sim_{\mathcal{T}}(Out\_s_y, c_j)$ between two parameters in a CLM$^+$ is an element of the set $\{Exact, PlugIn, Subsume, Intersection\}$. The latter set aims at value the semantic connection between an output parameter $Out\_s_y \in \mathcal{T}$ of s_y and $c_j \in$ Input(S_{Ws}) with $Exact$ being the best and $Intersection$ being the worst.

Moreover, a CLM$^+$ aims at storing non-functional properties of web services as a vector in $\Re^n$. Therefore, any service s_y referred to in the matrix contains not only semantic connections with some other services of S_{Ws}, but also its own non-functional properties $\boldsymbol{q}_{s_y} \in \Re^n$.

Example 1 *(Illustration of the CLM$^+$ indexes and labels.)*
Let $\{S_i\}_{i,i\in\{1,...,6\}}$ *be the set of web services* S_{Ws} *(table 1). The number of rows and columns of the CLM$^+$ is equal to 6 according to definition 2. Thus rows, columns of the CLM$^+$* $\mathcal{M}$ *are indexed by* $\{1, ..., 6\}$ *and labelled by the concepts Language, French, English, Text, PhoneNum and AckSMS, respectively (table 2).* $\mathcal{M}$ *refers to a CLM$^+$ with entries in* $\mathcal{P}(S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re)$*. The non-functional properties of* $S_{i,1\leq i\leq 6}$ *refer to a simple cost value in* $\Re$*.*

The CLM$^+$ construction depends on the number of output and input parameters of web services in S_{Ws}. Suppose $\#(Output(S_{Ws}))$ and $\#(Input(S_{Ws}))$ be respectively the number of output parameters of services in S_{Ws} and the number of input parameters of services in S_{Ws}. The algorithmic complexity for the causal link matrix construction is $\theta(\#(Input(S_{Ws})) \times \#(Output(S_{Ws})))$ or $\theta((Max\{\#(Input(S_{Ws})),$

$\#(Output(S_{Ws}))\}^2)$ so square in the worst case [26]. In other words, the CLMs$^+$ construction consists of finding a semantic similarity $score$ between the output parameters of all web services $s_y \in S_{Ws}$ and the input parameters of another web service in S_{Ws}. In case $score$ is not null, the triple $(s_y, score, \boldsymbol{q}_{s_y})$ is added in the CLM$^+$ according to definition 2. For further details [26] defines the whole process of the CLM$^+$ construction.

Example 2 *(CLM$^+$ illustration)*
The entry $m_{4,4}$ (i.e., $m_{Text,Text}$) of the matrix is equal to $\{(S_1, \sqsubseteq, 1), (S_2, \sqsubseteq, 4)\}$. In S_{Ws} there is a service S_1 with an input parameters $Text$ and an output parameter $EnglishText$, which is semantically similar to $Text$. $\langle S_1, Sim_{\mathcal{T}}(EnglishText, Text), S_3\rangle$ is a valid causal link. The $EnglishText$ and $Text$ concepts match with the Plug-in match *($\sqsubseteq$ in the matrix) according to the definition of $Sim_{\mathcal{T}}$. In this way all causal links are referred in the CLM$^+$ $\mathcal{M}$ as follows ($\equiv$ refers to the $Exact$ match):*

$$\mathcal{M} = \begin{pmatrix} \emptyset\ \emptyset\ \emptyset & \{(S_1,\sqsubseteq,1)\} & \emptyset & \emptyset \\ \emptyset\ \emptyset\ \emptyset & \{(S_2,\sqsubseteq,4)\} & \emptyset & \emptyset \\ \emptyset\ \emptyset\ \emptyset & \{(S_1,\sqsubseteq,1),(S_2,\sqsubseteq,4)\} & \emptyset & \emptyset \\ \emptyset\ \emptyset\ \emptyset & \{(S_1,\sqsubseteq,1),(S_2,\sqsubseteq,4)\} & \emptyset & \{(S_3,\equiv,1),(S_4,\equiv,3)\} \\ \emptyset\ \emptyset\ \emptyset & \emptyset & \emptyset & \{(S_3,\equiv,1),(S_4,\equiv,3)\} \\ \emptyset\ \emptyset\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

The key contribution of CLM$^+$ is a formal and semantic model to represent and manage a relevant set of services together with their non-functional properties. Web services of S_{Ws} are discovered first, to facilitate the composition process. Therefore the set of web services S_{Ws} is closed in order to limit the dimension of CLM$^+$. Such a model enables performance analysis of the proposed compositions by considering causal links and non-functional properties of services. CLM$^+$ aims at pre-chaining web services according to their semantic similarity based on their Output/Input specification. CLM$^+$ describes all possible matchings between all the web services in S_{Ws} as semantic connections. Moreover, the CLM$^+$ model is an interesting trade-off to support development activities such as services composition verification (valid causal link) or repair, by insertion and deletion of web services in the compositions.

Once web services in S_{Ws} are semantically chained according to the causal link criteria, the composition algorithm proceeds by generating the compositions graph.

4.4 Web Service Composition Process

The actual web services composition is performed using a graph-based approach, starting from the service request outputs, and possible effects, and composing backwards in the direction of the service request inputs and possible preconditions. The composition algorithm is executed after performing service discovery and CLM$^+$ construction. The CLM$^+$ contains the services that match the service request goals, and have valid causal links. The algorithm aims at finding a set of services with exact interface matchings ($Exact$), but other semantic matchings ($PlugIn$, $Subsume$, $Intersection$), are also considered in the graph composition algorithm. This is a realistic approach, since perfect matches may not always be possible. The non-functional properties are taken into account to optimise the search for service compositions. If a graph composition branch

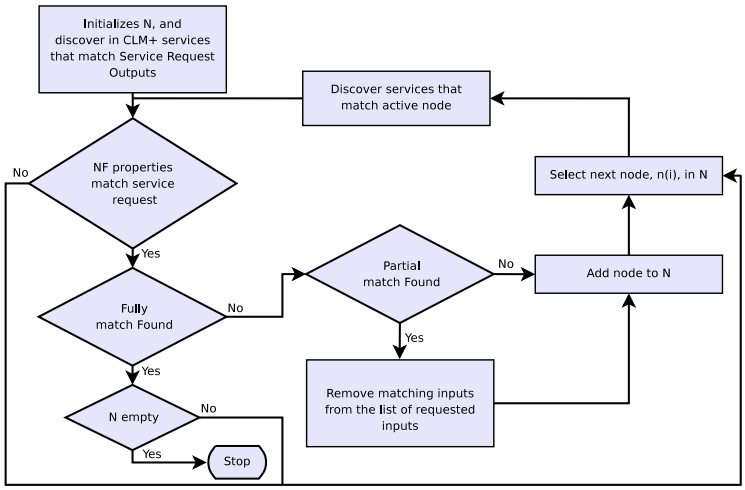

Fig. 4. Web Service Composition Algorithm

does not comply with the requested non-functional properties, the composition on this branch is aborted. Figure 4 depicts the graph-based service composition algorithm.

The algorithm defines N as the set of nodes to be resolved. Each element of N represents a node with inputs that do not fully match the inputs of the service request. The algorithm initializes N with the services that provide outputs $Out(s_0)$ from the original service request. After that, the algorithm evaluates whether the retrieved set of nodes require the same inputs $In(s_0)$ as the service request. If services that match both semantic descriptions $Out(s_0)$ and $In(s_0)$ are found, and N is empty, and the services satisfy the non-functional properties of service request, the graph composition algorithm stops. In case the query returns another service S_z that does not match $In(s_0)$, but delivers $Out(s_0)$ and matches the requested non-functional properties, S_z is added to N. The algorithm then processes each node n_i of N, by searching in the CLM$^+$ for services that match the unresolved n_i inputs (and possibly preconditions). For each matching service found, the composition graph is checked, inspecting whether the composition's aggregated non-functional properties match the requested non-functional properties, if it complies the service being resolved is removed from N. If there is no match, the composition graph branch that is being resolved is pruned, meaning that the elements being resolved are removed from N, and the composition branch is removed from the graph composition. Another heuristic that can be used to avoid unrealistic compositions is to limit the graph depth, restricting in this way the maximum number of services in a service composition.

Applying the graph composition algorithm to the running example, in the first step, services that provide as output an $AckSMS$, defined on the $TelecomOnt$ ontology, are selected. Two services $\{S_3, S_4\}$ are found in the CLM$^+$ matrix. None of these services fully match the service request inputs, but they provide an $Exact$ match to one of the requested inputs ($PhoneNum$), so in these branches this requested input is set as solved for the graph composition. Given that not all the inputs have been solved, and provid-

ing that the considered non-functional property $NFPOnt\#Cost$ is satisfied, services $\{S_3, S_4\}$ are stored in N as services that provide the requested output, with an $Exact$ semantic match, but do not completely match the requested input. In the second step, S_3 is resolved by discovering services in the CLM$^+$ that provide an output semantically related to the input of S_3, namely $Text$. Services $\{S_1, S_2\}$ have been discovered to provide the text message to S_3. These services resolve the requested inputs $In(s_0)$, although only as a partial semantic match ($Plugin$), and they meet the requested non-functional property, so that the search is closed on these branches. Having reached the $In(s_0)$ on these branches, N is inspected to check whether it is empty or not. N still contains S_4 to be resolved. In the third step, S_4 is resolved, also by using services $\{S_1, S_2\}$, and the composition process is finished for this branch. In this step, the aggregated non-functional property $Cost$ of the $S_2 \rightarrow S_4$ does not meet non-functional property requirement of the service request, so this graph branch is removed from the composition graph. After this step, N is checked, and since it is empty the algorithm stops. Table 3 represents the steps discussed above, the compositions found by the graph composition algorithm, and their respective aggregated non-functional properties.

Table 3. Service Compositions

Step	N	Compositions	NF properties
1	$\{S_3, S_4\}$	S_3	1
		S_4	3
2	$\{S_4\}$	$S_1 \mapsto S_3$	2
		$S_2 \mapsto S_3$	5
		S_4	3
3	{-}	$S_1 \mapsto S_3$	2
		$S_2 \mapsto S_3$	5
		$S_1 \mapsto S_4$	4

Table 3 shows that the service developer obtains three alternative compositions. Further computations could be done to reduce these possibilities and determine the "best" composition as a function of the measured semantic similarity and the non-functional properties. However, we believe that the service developer should receive all found compositions that match his request, and choose the one(s) that best fit his needs himself. Nevertheless, we suggest in the sequel an algorithm to rank the generated compositions, using the compositions semantic similarity and non-functional properties values.

4.5 Ranking of Composition Results

Our web service composition algorithm aims at retrieving compositions with valid causal links and also ensuring that the non-functional properties of the service request are satisfied by the generated compositions. However, our algorithm may return more than one composition, since some services can satisfy the same goals with different non-functional properties, or can satisfy *semantically close* goals with the same non-functional properties. In order to help service developers in their choice of service composition, we propose to rank composition results, for example, by first considering the semantic value of their causal links and after that, using the end-to-end non-functional

properties of the composite services, in case the compositions have identical causal link values. To this end we assign a score for each kind of semantic connection. A causal link with an $Exact$ matching is valued to 1, a causal link with a $PlugIn$ matching is valued to $\frac{3}{4}$, a causal link with a $Subsume$ matching is valued to $\frac{1}{2}$ and a causal link with a $Intersection$ matching is valued to $\frac{1}{4}$. Such a valuation is consistent since an $Exact$ matching between an output parameter and an input parameter is more preferred than a causal link with a $PlugIn$, $SubSume$ or $Intersection$ matching.

Algorithm 1: Ranking of Composition Results.

```
1  Input: An unordered set of composition results {S_{c_1}, ..., S_{c_n}}.
2  Result: An ordered set of composition results (based first on causal links and second on
           non functional properties of services).
3  begin
4      foreach S_{c_i} do
5          semantic_quality_S_{c_i} ← Average of causal links in S_{c_i};
6          NF_quality_S_{c_i} ← Function of NF properties in S_{c_i};
7      end
8      ({S_{c_1}, ..., S_{c_n}}, ≤) ← Ordering {S_{c_1}, ..., S_{c_n}} first by means of their
       semantic_quality and then by means of their NF_quality;
9      return ({S_{c_1}, ..., S_{c_n}}, ≤);
10 end
```

Non-functional properties of compositions are required in case two potential compositions of web services S_{c_i} and S_{c_j} have the same semantic quality. We overcome this issue by valuing each composition result S_{c_i} by means of a function (line 6 of algorithm 1) of the non-functional properties involved in S_{c_i}. The latter function depends on the non-functional properties of the atomic services of the composition. For instance, a sum is required to value the final cost of a composite service whereas the minimum is required to compute the throughput of a composite service. Since web services may have multiple non-functional properties, it is necessary to weight these properties, e.g., by means of user preferences. For example, an end-user may give more importance to the cost of a composite service whereas an another end-user may prefer the composite web service with the best throughput. In the service developer scenario such a ranking method could help the developer especially in case a large amount of valid composition results are returned.

5 Related Work

Recently the authors of [27] have addressed in detail the problem of interleaving web service discovery and composition, but have considered only simple workflows where web services have one input and one output parameter. In this case the web service composition plan is restricted to a sequence of limited web services corresponding to a linear workflow of web services. The suggested solution retrieves a sequence of causal links between web services, hence a linear and total order of services. Aiming of generating a composite service plan out of existing services, in [28] a composition path is proposed that consists of a sequence of operators that compute data, and connectors

that provide data transport between the operators. The search for possible operators to construct a sequence is based on the shortest path algorithm on the graph of the operators space. However, only two kinds of services (operator and connector) with one input and one output parameter are considered, which means that only the simplest case of service composition is covered. Contrary to [27] and [28], the model proposed in this paper may also consider services with more than one input and output parameter.

In [29], a composition of services is considered as a directed graph, where nodes are linked by the matching compatibility ($Exact$, $Subsume$, $PlugIn$, $Disjoint$) between input and output parameters. Based on this graph, the shortest sequence of web services from the initial requirements to the goal can be determined. This sequence corresponds to an ordered set of web services, so that this set matches all expected output parameters given the inputs provided by a user. [14] perform semantic web service composition by pre-computing the causal link matrix. Their composition strategy based on AI planning performs a regression-based approach and returns a set of correct, complete and consistent plans in which services are actions semantically linked by causal links. However, these two approaches [29, 14] compute the best composition according to the semantic similarity of output and input parameters of web services, without considering any non-functional properties of these services. A formalism and modelling tool called interface automata has been introduced in [30] to represent web services and perform compositions. Atomic services are stored as a graph where each node represents input and output parameters and edges represent web services. Each web service contains a description of its inputs, outputs, and dependencies of other web services. Web service descriptions and the graph are used to discover composition results that satisfy a service request. In case several alternative compositions are found, no optimization mechanism for selection is provided, so that in case several composition results match a request the most suitable compositions still have to be selected.

In [31] a composer is introduced to perform web services composition. The composer supports the end user to select web services for each activity in the composition and to create flow specifications to link them. Upon selecting a web service, the web services that can produce an output that could be fed as the input of the selected service are listed, after filtering based on profile descriptions. The user can manually select the service that he wants to fit in at a particular activity. After selecting all the services, the system generates a composite process in DAML-S. The composition is executed by calling each service separately, and passing the results between services according to the flow specifications. However, the composition is still semi-automatic because the user must select a web service in a restricted list. Our formal model presented in this paper aims at automating the process of web service selection according to the causal link criterion and the non-functional properties of services.

6 Final Remarks

Although web services technology is still in its infancy, some proposals are being made to enable dynamic composition of web services. Nevertheless, to the best of our knowledge, few of these proposals address both functional and non-functional properties of web services to optimize the composition process. In this paper we outlined the main

challenges faced in semantic web services, i.e., dynamic composition and optimization based on non-functional properties. To this end we described a framework for the functional composition of web services. Starting from a service developer service request, we successively apply web service discovery, causal link matrix computation, web service composition and optimization based on non-functional properties of services. By computing a causal link matrix, we ensure that the obtained compositions have valid semantic connections between component web services. Finally, the set of valid service compositions is selected by considering the non-functional properties of web services involved in the composition. If a composition does not match the non-functional properties of the service request, it is neglected. Our composition approach is quite general and can be easily applied to web services described using OWL-S (service profile), WSMO (capability model) or SA-WSDL specification.

In future work, we intend to investigate how an approach based on process aspects can be combined with the approach reported in this paper. This work should allow more composition problems to be solved, increase the number of valid composition results and improve the correctness of the composition process.

Acknowledgments. This work is supported by the European IST SPICE project (IST-027617) and the Dutch Freeband A-MUSE project (BSIK 03025).

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web services: concepts, architectures and applications. Springer-Verlag (2004)
2. Brodie, M.L., Bussler, C., de Bruijn, J., Fahringer, T., Fensel, D., Hepp, M., Lausen, H., Roman, D., Strang, T., Werthner, H., Zaremba, M.: Semantically enabled service-oriented architectures: a manifesto and a paradigm shift in computer science. Technical report, DERI (2005)
3. Sycara, K.P., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. J. Web. Sem. **1**(1) (2003) 27–46
4. Constantinescu, I., Faltings, B.: Efficient matchmaking and directory services. In: IEEE/WIC International Conference on Web Intelligence. (2003) 75–81
5. Constantinescu, I., Faltings, B., Binder, W.: Type based service composition. In: 13th International World Wide Web Conference (Alternate track papers & posters). (2004) 268–269
6. Klusch, M., Fries, B., Khalid, M., Sycara, K.: OWLS-MX: Hybrid OWL-S service matchmaking. In: First International Symposium on Agents and the Semantic Web. (2005)
7. Paolucci, M., Sycara, K.P., Kawamura, T.: Delivering semantic web services. In: 12th International World Wide Web Conference (Alternate paper tracks). (2003) 829–837
8. Sirin, E., Parsia, B., Hendler, J.A.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intelligent Systems **19**(4) (2004) 42–49
9. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: ICSOC 2003. LNCS, vol. 2910, Springer (2003) 43–58
10. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: 12th International World Wide Web Conference, ACM Press (2003) 403–410

11. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: 11th International World Wide Web Conference, ACM Press (2002) 77–88
12. Pistore, M., Roberti, P., Traverso, P.: Process-level composition of executable web services: ”on-the-fly” versus ”once-for-all” composition. In: ESWC 2005. LNCS, vol. 3532, Springer (2005) 62–77
13. Bertoli, P., Hoffmann, J., Lécué, F., Pistore, M.: Integrating discovery and automated composition: from semantic requirements to executable code. In: IEEE International Conference on Web Services. (2007) 815–822
14. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: ISWC 2006. LNCS, vol. 4273 (2006) 385–398
15. Cordier, C., Carrez, F., van Kranenburg, H., Licciardi, C., van der Meer, J., Spedalieri, A., Rouzic, J.P.L.: Addressing the challenges of beyond 3G service delivery: the SPICE platform. In: 6th International Workshop on Applications and Services in Wireless Networks. (2006)
16. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.1 (2003)
17. Smith, M.K., McGuiness, D., Volz, R., Welty, C.: Web Ontology Language (OWL) guide, version 1.0. W3C. (2002)
18. Ankolenkar, A., Paolucci, M., Srinivasan, N., Sycara, K.: OWL Web Ontology Language guide. W3C. (2004)
19. Fensel, D., Kifer, M., de Bruijn, J., Domingue, J.: Web service modeling ontology (WSMO), W3C member submission (2005)
20. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web services standards. In: 1st International Conference on Web Services. (2003) 395–401
21. Küsters, R.: Non-Standard Inferences in Description Logics. LNCS, vol. 2100. Springer (2001)
22. Russell, S., Norvig, P.: Artificial Intelligence: a modern approach. Prentice-Hall (1995)
23. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: ICWS 2002. LNCS, vol. 2342, Springer (2002) 333–347
24. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: 12th International World Wide Web Conference, ACM Press (2003) 331–339
25. Lécué, F., Delteil, A., Léger, A.: Applying abduction in semantic web service composition. In: IEEE International Conference on Web Services. (2007) 94–101
26. Lécué, F., Léger, A.: Semantic web service composition through a matchmaking of domain. In: 4th IEEE European Conference on Web Services. (2006) 171–180
27. Lassila, O., Dixit, S.: Interleaving discovery and composition for simple workfows. In: First International Semantic Web Services Symposium. (2004)
28. Mao, Z.M., Katz, R.H., Brewer, E.A.: Fault-tolerant, scalable, wide-area internet service composition. Technical Report CSD-01-1129, University of California at Berkeley (2001)
29. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: 1st International Conference on Web Services. (2003) 38–41
30. Alfaro, L.D., Henzinger, T.A.: Interface automata. In: 8th European software engineering conference / 9th ACM SIGSOFT international symposium on Foundations of software engineering. (2001) 109–120
31. Sirin, E., Hendler, J.A., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: 1st Workshop on Web Services: Modeling, Architecture and Infrastructure. (2003) 17–24