Fully Automated Web Services Orchestration in a Resource Retrieval Scenario

Azzurra Ragone[†], Tommaso Di Noia[†], Eugenio Di Sciascio[†],
Francesco M. Donini[‡], Simona Colucci[†]

Politecnico di Bari[†]
Via Re David, 200, I-70125, Bari, Italy
{a.ragone,t.dinoia,disciascio,s.colucci}@poliba.it

Università della Tuscia[‡]
via San Carlo, 32, I-01100, Viterbo, Italy
donini@unitus.it

Abstract

We propose a framework and polynomial algorithms for semantic-based automated Web service orchestration, fully compliant with Semantic Web technologies. The approach exploits the recently proposed Concept Abduction inference service in Description Logics to solve Concept Covering problems. We present how the proposed approach deals with not exact solutions, computing an approximate orchestration with respect to an agent request modeled using a significant subset of OWL-DL.

1 Introduction

Web service composition amounts to the orchestration of a certain number of existing web services to provide a composite service able to satisfy the user's requirements, in case a single web service is not adequate. Various approaches have been recently proposed able to provide the orchestration of a web services set, based on different aspects of both web service [19, 14, 26] and composition flow [31, 29, 33, 25] modeling. But, due to the lack of automatic procedures able to cope with non-exact matches, in many of such approaches there is the need of a human intervention in order to establish candidate execution flows. Fully-automation of the web services discovery and orchestration requires as a first step providing service descriptions – what a service offers– that have to be well defined, machine understandable and processable. The Semantic Web initiative aims to provide web resources with a meaning with the aid of formal languages and ontologies to allow reasoning on service description. The approach we propose here models the orchestration with respect to a resource retrieval scenario, that is, a scenario where there is a request and several available resources, which can satisfies the request in a one to one match (one resource retrieved for the request) or in a one to many match (several retrieved resources whose combination satisfy the request). Our aim is to model a user oriented and friendly framework where a typical request is like *"I'd like to book a hotel provided with a swimming pool and a fitness center*" rather than only "Effects = `HOTEL_RESERVATION`" and a typical service description is like "We book for you hotels near the sea provided with all the facilities: swimming pool, fitness center, children area and restaurants" rather than only "Preconditions = `VALID_CREDIT_CARD`; Effects = `HOTEL_RESERVATION`". What we would like to point out is that limiting the information provided by services to only preconditions (and / or inputs) and effects (and / or outputs) makes hard to detail the service provided. We believe that a service should provide two kinds of information separately: (1) description of the service behavior, even not a functional one, (2) execution information.

In this paper we propose a framework and algorithms, fully compliant with the Semantic Web vision and its related technologies, for the automated discovery and composition in a semantic web-services orchestration scenario. To this aim we exploit the recently proposed Concept Abduction [16] inference service in Description Logic to generalize and extend Concept Covering[24]. We devise a greedy algorithm for Concept Covering in a subset of OWL-DL and exploit it for semantic service discovery. Our framework also allows to compose a flow of discovered services, *i.e.*, an orchestration, taking into account both effects duplication and approximate solutions computed in polynomial time.

The remaining of the paper is structured as follows: next section describes the subset of OWL-DL we concentrate on; then we present our extension of Concept Covering definition and a greedy algorithm, which uses Concept Abduction, to determine a Concept Cover. In Section 3 we exploit the previously defined Concept Covering algorithm in a general web-service framework to carry out a semantic web-services orchestration. The approach and behavior of the related algorithms are thoroughly explained with the aid

of an example in Section 4 and Section 5. Section 6 is devoted to the description of a prototype implementing the approach. Section 7 summarizes relevant related work, while last Section draws the conclusions and outlines future research directions.

2 $\mathcal{ALN}$ Description Logic and OWL

Due to the lack of space, in what follows we assume the reader is a bit familiar with Description Logics (DLs)[2] syntax and semantics. An ontology can be modeled in a DL using a set of *inclusion assertions*, and *definitions*, which impose restrictions on possible interpretations according to the knowledge elicited for a given domain or task. Historically, sets of such inclusions are called TBox (Terminological Box) $\mathcal{T}$. The basic reasoning problems for concepts in a DL are satisfiability, which accounts for the internal coherency of the description of a concept (no contradictory properties are present), and subsumption, which accounts for the more general/more specific relation among concepts, that forms the basis of a taxonomy. Different Description Logics can be identified depending on the allowed constructors and consequently their expressiveness. In the rest of the paper we refer to $\mathcal{ALN}$(**A**ttributive **L**anguage with unqualified **N**umber restrictions). We use a $simple - TBox$ in order to express the relations among objects in the domain. A $simple - TBox$ allows only axioms (for both inclusion and definition) where the left side is represented by a concept name.

Ontologies using the above logic can be easily modeled using languages for the Semantic Web[28][1]. The strong relations between Description Logics and the above introduced languages for the Semantic Web [3] is also evident in the definition of the three OWL sub-languages. *OWL-Lite*: It allows class hierarchy and simple constraints on relation between classes. *OWL-DL*: Based on Description Logics theoretical studies, it allows a great expressiveness keeping computational completeness and decidability. *OWL-Full*: Using such a language, there is a huge syntactic flexibility and expressiveness. This freedom is paid in terms of no computational guarantee.

In the rest of the paper we will use DL syntax instead of OWL-DL syntax, to make expressions much more compact.

3 Concept Covering via Concept Abduction

Standard inference services in DLs include subsumption and satisfiability. These are enough when a yes/no answer is needed. However there are scenarios that require explanation. In [16] the Concept Abduction Problem (CAP) was introduced and defined as a non standard inference problem for DLs, to provide an explanation when subsumption does not hold.

Definition 1 *Let C, D, be two concepts in a Description Logic $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms, where both C and D are satisfiable in $\mathcal{T}$. A* Concept Abduction Problem *(CAP), denoted as $\langle \mathcal{L}, C, D, \mathcal{T} \rangle$, is finding a concept H such that $\mathcal{T} \not\models C \sqcap H \equiv \bot$, and $\mathcal{T} \models C \sqcap H \sqsubseteq D$.*

Given a CAP, if H is a conjunction of concepts and no subconjunction of concepts in H is a solution to the CAP, then H **is an irreducible solution**. In [16] also minimality criteria for H and a polynomial algorithm to find solutions which are irreducible, for an $\mathcal{ALN}$ DL, have been proposed. How can a solution to a CAP be useful in a resource retrieval scenario? H, in a CAP solution, is the missing part in the available resource C needed to completely satisfy a request D. If there is a non-exact match it is because of H. H is what is not covered by C with respect to D. Based on this last remark we use solutions to sets of CAP to solve Concept Covering Problems as shown in the following.

In [24] the *best covering problem* in DLs was introduced as "...a new instance of the problem of rewriting concepts using terminologies", and proposed in [5] to cope with automated semantic web service composition. Unfortunately, the non standard inference services the authors exploited in order to define the concept covering, can be used only for DLs less expressive than $\mathcal{ALN}$(for a complete description see [35]).

We propose an extension to the previous definition of a Concept Covering, both eliminating limitations on the DL to be used and rewriting it in terms of Concept Abduction.

Definition 2 *Let D be a concept, $\mathcal{R} = \{S_1, S_2, ..., S_k\}$ be a set of concepts, and $\mathcal{T}$ be a set of axioms, all in a DL $\mathcal{L}$, where D and $S_1, \ldots, S_k$ are satisfiable in $\mathcal{T}$. The* Concept Covering Problem *(CCoP) for $\mathcal{V}$=$\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$ is finding a pair $\langle \mathcal{R}_c, H \rangle$ such that*
1. $\mathcal{R}_c \subseteq \mathcal{R}$, and the conjunction of concepts in $\mathcal{R}_c$, $C = \sqcap_{S \in \mathcal{R}_c} S$ is satisfiable in $\mathcal{T}$;
2. $H \in SOL(\langle \mathcal{L}, C, D, \mathcal{T} \rangle)$, and $\mathcal{T} \not\models H \sqsubseteq D$.
We call $\langle \mathcal{R}_c, H \rangle$ a solution *for $\mathcal{V}$, and say that $\mathcal{R}_c$* (partially) **covers** *D. Finally, we denote $SOLCCoP(\mathcal{V})$ the set of all solutions to a CCoP $\mathcal{V}$.*

Intuitively, $\mathcal{R}_c$ is the set of concepts that partially cover D w.r.t. $\mathcal{T}$, while the abduced concept H covers what is still in D and is not covered by C. There can be several solutions for a single CCoP, depending also on the strategy adopted for choosing concepts in $\mathcal{R}_c$. However, observe that – differently from the standard Set Covering Problem – a complete cover may not exist. Hence, minimizing the cardinality of $\mathcal{R}_c$ is not the aim of a CCoP; the aim is maximizing the covering, hence minimizing H. This argument led us to the definition of *best cover* and *exact cover* (see

[1] http://www.w3.org/TR/owl-features/

[15] for details). In [15] a tractable greedy algorithm is presented to solve a CCoP $\mathcal{V}$ in $\mathcal{ALN}$[2]. In the rest of the paper we will refer to it as $solveCCoP(\mathcal{R}, D, \mathcal{T})$. It takes as inputs a set of concepts $\mathcal{R}$ (a set of web service descriptions) a concept D (the request) and the reference ontology (the domain model) and returns a solution to the related $\mathcal{V}$.

4 Semantic Web Service Orchestration

We now show how the previously presented non-standard inference services can be used for service orchestration, *i.e.*, *discovery* and *composition*. Let us point out that we refer to a general framework, compliant with all the ones modeling Semantic Web Services with respect to a *Input-Output-Precondition-Effect* model. This implies that our approach can be reformulated using OWL-S [3] model with practically no changes.

As stated above, we believe that the service discovery process is a sub-problem of the more generic resource retrieval one. Having a request and several available resources potentially matching it, if it is not possible to retrieve some resources so that they completely satisfy the request, is an approximate solution possible? Is it possible to have explanation on such approximation? To explain and motivate the approach, we start with a simple semantic web-service model and then add features, enriching the model. In the initial model we define both the request D and the description WS_D of each web service as DL concept descriptions w.r.t. an ontology $\mathcal{T}$. A classification schema such as UNSPSC can then be used for tasks and related ontologies [13]. We assume the existence of a registry repository where service providers store, for each service, both all the information needed for the invocation and the description WS_D of the service itself.

Given a request D modeled w.r.t. an ontology $\mathcal{T}$, the steps needed in order to obtain a set of services satisfying as much as possible D are hence the following:

1. query the registry in order to obtain all the service descriptions WS_D which refers to the same $\mathcal{T}$.

2. put all the retrieved services description in a set $\mathcal{R}$.

3. call $solveCCoP(\mathcal{R}, D, \mathcal{T})$.

4. with reference to $\langle \mathcal{R}_c, H \rangle$ returned by $solveCCoP$ in the previous step, propose to the requester both H and, for each $WS_D \in \mathcal{R}_c$, the invocation information of the corresponding service.

$\mathcal{R}_c$ and H are, respectively, the set of resources representing an approximate solution to the retrieval problem and an explanation on why the solution is not an exact one.

[2] It is well known that the general set covering problem is NP-hard, then the computed solution will not be optimal in general (neither a full cover, nor a best one).

[3] http://www.daml.org/services/owl-s/1.0/owl-s.html

Using the above approach, a **discovery** is possible for the services available in the registry, which can be composed in order to satisfy D as far as possible.

Obviously, modeling the **composition** of the discovered services in the retrieved set requires taking into account also their execution information, *i.e.*, inputs, outputs, preconditions and effects specification. Without loss of generality, in what follows we will consider only preconditions and effects, as it is straightforward to extend the approach also considering inputs and outputs.

4.1 Precondition and Effects for Web Service Orchestration

In order to execute a web service, its preconditions must be satisfied, possibly using information provided by other web services. Moreover care has to be paid in avoiding the duplication of effects when composing services, which might be due to entailment relationships among different effects provided by services being composed.

Turning to the classical example proposed in [22], an agent booking organizing a trip and composing two services, one able to book both *a hotel stay and a flight* and another a *flight and a car rental*, would not be much appreciated if its outcome is two flights booked for the same trip, together with the hotel and the car.

In order to deal with the execution information, we extend the web service scenario model and define:

Request: *a pair $\langle D, P_0 \rangle$, where D is the description the requested service and P_0 are the preconditions provided with the request.*

For example, if the request is performed by a personal user agent, then it is able to provide some initial information to be used as preconditions for a service execution, *e.g.*, a valid credit card.

Web Service [4] : *a triple $\langle WS_D, P, E \rangle$, where WS_D represents provided service description, P the preconditions and E the effects.*

WS_D is a description of *the offered service*. Using a language endowed with a well-defined syntax and semantics, it models *what* the service offers. P and E are respectively the preconditions and the effects specification.

In our setting, WS_D and D are formulated as DLs concepts w.r.t. the domain ontology $\mathcal{T}_D$. For the sake of simplicity, here P_0,P and E are modeled as conjunction of atomic concepts represented in the Precondition/Effect on-

[4] A web service implemented by combining other web services is referred as *composite* "[...] to distinguish it form the ones implemented through conventional programming languages and invoking conventional services which are called basic"[1].Since a service being basic as composite is transparent to the client, in the following we address them as *web service* for simplicity.

tology $\mathcal{T}_{P/E}$. This is a simple TBox with no role and containing only inclusion axioms.

A simple covering solution, as the one proposed above, cannot deal with the P, E specifications of the services. To compose web services dealing with their P/E, we introduce an informal definition of **web service flow** with respect to some initial preconditions P_0, from now on $\mathcal{WSF}(P_0)$.

It is a finite sequence of web services $(ws_1, ws_2, ...ws_i, ..., ws_n)$, where for each web service ws_i belonging to to the flow, all the following conditions hold:

1. The preconditions of the first web service ws_1 executed in a composite web service are satisfied by the information P_0 provided with the initial request.

2. In order to execute a web service ws_i, the available information for preconditions satisfaction comes from the effects of all the web services ws_j, with $j < i$, previously executed.

3. Effects duplication after the execution of the composite web service has to be avoided.

The available information for ws_i is the conjunction of both the effects produced by $ws_j \in \mathcal{CWS}(P_0)$, with $j < i$, along the execution flow and the initial preconditions P_0.

In a more formal way, indicating with AI_i the available information for ws_i and with E_j the effects produced by ws_j, with $j < i$, the following relation ensues:

$$AI_i = P_0 \sqcap E_1 \sqcap E_2 \sqcap ... \sqcap E_{i-1}$$

Now we can formally define a **web service flow**.

Definition 3 *A **web service flow** with respect to some initial preconditions P_0 is a finite sequence of web services $\mathcal{WSF}(P_0) = (ws_1, ws_2, ...ws_i, ..., ws_n)$ with $i = 1..n$, where for each web service $ws_i \in \mathcal{WSF}(P_0)$ all the following conditions hold:*

1. for ws_1, $P_0 \sqsubseteq P_1$.

2. for ws_i, with $i > 1$, $AI_i \sqsubseteq P_i$.

3. for ws_i, with $i > 1$, for each concept name A occurring in E_i, $AI_i \not\sqsubseteq A$.

We indicate with $\mathcal{D}_{\mathcal{WSF}}$, the set of web service descriptions in $\mathcal{WSF}(P_0)$. $\mathcal{D}_{\mathcal{WSF}} = \{WS_{Di} | ws_i \in \mathcal{WSF}(P_0)\}$.

Based on the definition of **web service flow**, here we define a **composite web service** with respect to a request.

Definition 4 *Let $\mathcal{R} = \{\langle WS_{Di}, P_i, E_i\rangle\}$, with i=1..k, be a set of web services ws_i, and $\langle D, P_0\rangle$ be a request, such that both D and WS_{Di} are modeled as concept descriptions in a DL w.r.t. a domain ontology $\mathcal{T}_D$, and P_0, P_i and E_i modeled as conjunction of concept names w.r.t. a Precondition/Effect ontology $\mathcal{T}_{P/E}$.*

*A **composite web service** for $\langle D, P_0\rangle$ with respect to $\mathcal{R}$, $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$, is a web service flow such that for each ws_j in the execution flow, $\mathcal{D}_{\mathcal{CWS}(\langle D,P_0,\mathcal{R}\rangle)} = \{WS_{Dj} | ws_j \in \mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)\}$, covers D.*

A composite web service is an execution flow such that it can be started using some information provided by the requester (P_0) as initial preconditions, and the provided service covers the user request description (D).

4.2 Computing a Composite Web Service

We now adapt *solveCCoP* to cope with web service preconditions and effects and present an algorithm (Figure 1) to automatically compute a composite web service.

For such purpose the definition of *executable web service* and *executable set* are useful.

Definition 5 *Given a web service flow $\mathcal{WSF}(P_0) = (ws_1, ws_2, ..., ws_n)$, we say that a web service is an **executable web service** ws^{ex} for $\mathcal{WSF}(P_0)$ if and only if*

1. $ws^{ex} \notin \mathcal{WSF}(P_0)$.

2. $\mathcal{WSF}'(P_0) = (ws_1, ws_2, ..., ws_n, ws^{ex})$ is a web service flow.

An **executable web service** ws^{ex} for $\mathcal{WSF}(P_0)$ is a web service which can be invoked after the execution of $\mathcal{WSF}(P_0)$, *i.e.*, its preconditions are satisfied after the execution of $\mathcal{WSF}(P_0)$, and such that its effects are not already provided by $\mathcal{WSF}(P_0)$.

Definition 6 *Given a web service flow $\mathcal{WSF}(P_0)$ and a set of web services $\mathcal{R} = \{ws_i\}$ we call* executable set *for $\mathcal{WSF}(P_0)$, the set of all the $ws_i \in \mathcal{R}$ such that ws_i is an executable service for $\mathcal{WSF}(P_0)$}.*
$\mathcal{EX}_{\mathcal{WSF}(P_0)} = \{ws_i^{ex} | ws_i^{ex}$ is an executable service for $\mathcal{WSF}(P_0)\}$

The *executable set* is hence the set of all the services that can be invoked after the execution of a web service flow. In $\oslash$ a call to *rankPotential(WS_{Di},$D_{uncovered}$)* (see [18]) establishes the order relation. The values returned by *serviceComposer* represent the composite web service $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$ and the uncovered part, $D_{uncovered}$, of the request description D. Notice that since *serviceComposer* uses a greedy approach, $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$ represents the quickest solution to the orchestration problem, as well as since $D_{uncovered}$ is computed via CAP, it is not "the" explanation for the uncovered part of the demand, but is "an" explanation for that.

5 Algorithm behavior

In order to describe the *serviceComposer* behavior we model a simple travel reservation scenario. The request description D and the web service descriptions WS_{Di} are modeled with respect to the toy ontology $\mathcal{T}$ pictured in Figure 2. As in the prototype system (see Section 6) we implemented *findIrred*[16] to solve Concept Abduction Problems

Algorithm $serviceComposer(\mathcal{R}, \langle D, P_0\rangle, \mathcal{T})$
input a set of services $\mathcal{R} = \{ws_i = \langle WS_{Di}, P_i, E_i\rangle\}$, a request $\langle D, P_0\rangle$, where D and WS_{Di} are satisfiable in $\mathcal{T}$
output $\langle \mathcal{CWS}, H\rangle$
begin algorithm
$\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle) = \emptyset$;
$D_{uncovered} = D$;
$H_{min} = D$;
do
compute $\mathcal{EX}_{\mathcal{CWS}(\langle D,P_0,\mathcal{R}\rangle)}$;
$WS_{Dmin} = \top$;
for each $ws_i \in \mathcal{EX}_{\mathcal{CWS}(\langle D,P_0,\mathcal{R}\rangle)}$
if $\mathcal{D}_{\mathcal{CWS}(\langle D,P_0,\mathcal{R}\rangle)} \cup \{WS_{Di}\}$ *covers* $D_{uncovered}$ **then** /* $\otimes$ */
$H = solveCAP(\langle \mathcal{L}, WS_{Di}, D_{uncovered}, \mathcal{T}\rangle)$; /* $\odot$ */
/ Choose S_i based on an order */*
if $H \prec H_{min}$ **then** /* $\oslash$ */
$WS_{Dmin} = WS_{Di}$;
$H_{min} = H$;
end if
end if
end for each
if $WS_{Dmin} \not\equiv \top$ **then**
$\mathcal{R} = \mathcal{R} \backslash \{ws_i\}$;
$\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle) = (\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle), ws_i)$;
$D_{uncovered} = H_{min}$;
end if
while$(WS_{Dmin} \not\equiv \top)$;
return $\langle \mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle), D_{uncovered}\rangle$;
end algorithm

Figure 1. The algorithm *serviceComposer*

($\odot$ in *serviceComposer*), the results presented below refer to such algorithm and *rankPotential*.

We write H_{WSname} to express that H is a solution to the CAP $\langle \mathcal{L}, WSname_D, D_{uncovered}, \mathcal{T}\rangle$, computed using $findIrred$ and $|H_{WSname}|$ to denote the *length* of such solution computed using $rankPotential$.

Let us suppose the registry $\mathcal{R}$ stores the following web services, whose description is in accordance with the ontology in figure 2.

$\mathcal{R} = \{MyCar, SuperCar, HotelBooker, NiceHolidays, SunnyHouse\}$

MyCar is a car rental service for the USA. It has agreements with both Flight companies and Hotel reservation networks; it produces a CAR_RESERVATION only if you provide also FLIGHT_RESERVATION and HOTEL_RESERVATION with a VALID_CREDIT_CARD

$\langle MyCar_D$ = ∃carRenting ⊓ ∀carRenting.
(∃allowedCountries ⊓ ∀allowedCountries.USA),
P = VALID_CREDIT_CARD ⊓ FLIGHT_RESERVATION⊓
HOTEL_RESERVATION,
E = CAR_RESERVATION⟩

SuperCar is a car rental service for North America, the renting price includes the insurance. It has offices located only in North America airports, it produces a CAR_RESERVATION only if you provide also FLIGHT_RESERVATION with a VALID_CREDIT_CARD.

USA ⊑ NorthAmerica | Canada ⊑ NorthAmerica

NorthAmerica ⊑ Country | WesternEurope ⊑ Country |
NorthAmerica ⊑ ¬WesternEurope

Sea ⊑ Location | Mountain ⊑ Location | Sea ⊑ ¬Mountain

SwimmingPool ⊑ Facilities | FitnessCenter ⊑ Facilities

BedRoom ⊑ ∃hasBeds | DoubleRoom ≡ BedRoom ⊓ (= 2 hasBeds) |
SingleRoom ≡ BedRoom ⊓ (= 1 hasBeds) |
WeddingRoom ≡ DoubleRoom ⊓ ∀hasBeds.DoubleBed

Figure 2. The reference ontology for the example

$\langle SuperCar_D$ = ∃carRenting ⊓ ∀carRenting.
(∃priceIncludes ⊓ ∀priceIncludes.Insurance⊓
∃allowedCountries⊓
∀allowedCountries.NorthAmerica),
P = VALID_CREDIT_CARD ⊓ FLIGHT_RESERVATION,
E = CAR_RESERVATION⟩

HotelBooker allows to reserve quality hotels with fitness centers and located near the sea. It has agreements with Flight companies, thus in order to obtain a HOTEL_RESERVATION you must provide a VALID_CREDIT_CARD and aFLIGHT_RESERVATION.

$\langle HotelBooker$ = ∃hotelReservation⊓
∀hotelReservation.(∃hasFacilities⊓
∀hasFacilities.FitnessCenter⊓
∃hasLocation ⊓ ∀hasLocation.Sea),
P = VALID_CREDIT_CARD ⊓ FLIGHT_RESERVATION,
E = HOTEL_RESERVATION⟩

NiceHolidays provides a holiday service. If you have a VALID_CREDIT_CARD it provides both a FLIGHT_RESERVATION to flights to North America countries and a HOTEL_RESERVATION in hotels located on mountains and provided with a swimming pool.

$\langle NiceHolidays$ = ∃hotelReservation⊓
∀hotelReservation.(∃hasFacilities
⊓∀hasFacilities.SwimmingPool⊓
∃hasLocation ⊓ ∀hasLocation.Mountain)⊓
∀flightBooking.(∃allowedAirport⊓
∀allowedAirport.NorthAmerica)⊓
∃flightBooking,
P = VALID_CREDIT_CARD,
E = HOTEL_RESERVATION ⊓ FLIGHT_RESERVATION⟩

SunnyHouse provides a house rental service. The houses to be let are endowed with two wedding rooms. Giving a VALID_CREDIT_CARD you are able to get a HOUSE_RESERVATION.

$\langle SunnyHouse$ = ∃houseRenting⊓
∀houseRenting.(= 2 hasRooms)⊓
∀hasRooms.WeddingRoom,
P = VALID_CREDIT_CARD,
E = HOUSE_RESERVATION⟩

Imagine you plan your holiday visiting the USA. You want to book a flight, to have a hotel reservation in a hotel and rent a car. You provide a VALID_CREDIT_CARD, as information for web service accessing.

Your request is formalized as follows:

$\langle D$ = ∀flightBooking.(∀allowedAirport.USA)⊓
⊓∀carRenting.(∀allowedCountries.USA)⊓
∃hotelReservation,

P_0 = VALID_CREDIT_CARD$\rangle$

In order to compute a $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$ the first step is to evaluate $\mathcal{EX}_{\mathcal{CWS}(\langle D,P_0,\mathcal{R}\rangle)}$ with respect to an empty web service flow $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle) = \emptyset$ ($\mathcal{EX}_0$ for short). As $P_0 =$ VALID_CREDIT_CARD then

$$\mathcal{EX}_0 = \{NiceHolidays, SunnyHouse\}$$

At this initial step $D_{uncovered} = D$ then:
$H_{NiceHolidays} =$ ∀flightBooking.(∀allowedAirport.USA)⊓
⊓∀carRenting.(∀allowedCountries.USA)
$|H_{NiceHolidays}| = 4$

$H_{SunnyHouse} =$ ∀flightBooking.(∀allowedAirport.USA)⊓
⊓∀carRenting.(∀allowedCountries.USA)⊓
∃hotelReservation
$|H_{SunnyHouse}|$ =NOT COMPUTED

$|H_{SunnyHouse}|$ is not computed because of the relation $H_{SunnyHouse} \sqsubseteq D_{uncovered}$ and then it cannot belong to a covering solution (see the condition $\otimes$ in *serviceComposer*). At this point we have:

- $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle) = (NiceHolidays)$
- $AI_1 =$ VALID_CREDIT_CARD ⊓ HOTEL_RESERVATION⊓ FLIGHT_RESERVATION
- $D_{uncovered} =$ ∀flightBooking.(∀allowedAirport.USA) ⊓∀carRenting.(∀allowedCountries.USA)

With respect to AI_1, the new executable set $\mathcal{EX}_1$ is:

$$\mathcal{EX}_1 = \{MyCar, SuperCar, SunnyHouse\}$$

Notice that $HotelBooker$ is not in $\mathcal{EX}_1$ because the effects it provides are already in AI_1.
$H_{MyCar} =$ ∀flightBooking.(∀allowedAirport.USA)
$|H_{MyCar}| = 3$

$H_{SuperCar} =$ ∀flightBooking.(∀allowedAirport.USA) ⊓ ∀carRenting.(∀allowedCountries.USA)
$|H_{SuperCar}| = 4$

$H_{SunnyHouse} =$ ∀flightBooking.(∀allowedAirport.USA)
⊓∀carRenting.(∀allowedCountries.USA)
$|H_{SunnyHouse}|$ = NOT COMPUTED

$MyCar$ will be added to $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$ because of $|H_{MyCar}| < |H_{SuperCar}|$ obtaining:

- $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle) = (NiceHolidays, MyCar)$
- $AI_2 =$ VALID_CREDIT_CARD ⊓ HOTEL_RESERVATION⊓ FLIGHT_RESERVATION ⊓ CAR_RESERVATION
- $D_{uncovered} =$ ∀flightBooking.∀allowedAirport.USA)

The next executable set ($\mathcal{EX}_2$) is:

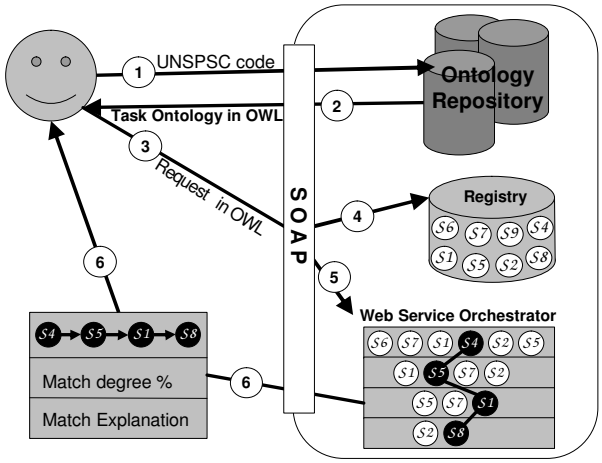

Figure 3. Schematic of the Web Services Orchestrator Architecture

$$\mathcal{EX} = \{SunnyHouse\}$$

At this point it should be easy to argue that $SunnyHouse$ cannot belong to the composite web service. The final result is then:

$\langle (NiceHolidays, MyCar),$
∀flightBooking.∀allowedAirport.USA$\rangle$

That is, with respect to the information provided by the user, the system is able to compute the candidate composite web service $(NiceHolidays, MyCar)$, but it cannot know if the flight service is specific for the USA. Nothing is specified about that in the description of selected web services. The missing information can be used to initiate a dialogue between the user and the orchestrator.

Notice that if you change your request description adding also as hotel location preference "near the sea" (∀hasLocation.Sea) then in the composite web service you will not get $NiceHolidays$ as it provides hotels near the mountains. It is noteworthy this does not mean that the less information the service provider uses the better for him/her (*e.g.*, you do not specify anything about the location); this is because if he/she underspecifies the service description it will cover less the user request description.

6 Prototype System

The proposed approach has been implemented in a prototype system, whose architecture is sketched in Figure 3.

In this version of the system we model each web service specification as an OWL-S 1.0 Profile instance. In order to deal with the web service semantic description D a new ObjectProperty $< owl : ObjectPropertyrdf : ID = "SemanticDescription" >$ was introduced, which

has cardinality restricted to 1, OWL-S $Profile$ class as domain and $< owl : Thing/ >$ as range, similarly to the $textDescription$ DatatypeProperty in OWL-S.

The communication with the system is carried out through SOAP messages. The main system components are:

•*Ontology Repository*. It contains task ontologies, each one referring to different UNSPSC family levels (as proposed in [13]).

•*Registry*. Here the web service specification are stored.

•*Web Services Orchestrator (WSO)*. It uses the $serviceComposer$ algorithm presented in Section 4.2. In order to solve Concept Abduction Problems, WSO uses MAMAS[5] implementing both $findIrred$ and $rankPotential$, endowed of an enhanced DIG[4] interface to support these non-standard inferences.

In what follows, we explain the system behavior with reference to Figure 3, where the system interaction is sketched with respect to a generic agent[6] request.

(1) In the initial step, the agent determines the task ontology that will be used (or the relative URL) selecting it via the UNSPSC code.

(2) The selected ontology (or the relative URL) is sent back to the agent.

(3) Using the selected ontology, the agent formulates a request using its OWL Classes and Properties. The request is then sent to the system together with the initial preconditions it is able to provide. At the current stage of the prototype, preconditions and effects are selected within a classification modeled using OWL Classes. From the information provided by the agent, the system selects two types of information: (a) the UNSPSC code identifying the ontology and (b) the OWL request description and the initial preconditions.

(4) Using (a) in step (3), the system retrieves the corresponding web services instances and sends them to WSO, with the corresponding task ontology.

(5) The request description and the initial preconditions selected in (b) part of step (3), are sent to WSO.

(6) After a conversion from OWL syntax to DIG one, WSO computes, with the aid of MAMAS, a composite service, the matching degree and in case of an approximate solution an explanation of which part of the request is not covered by the composite service (that is an explanation of why the covering is not complete).

[5]http://193.204.59.227:8080/MAMAS-devel

[6]Here agent is meant in the broader sense of the term, *i.e.*, an automated one or a human user

7 Related Work

Discovery of web-services described in DAML-S (the parent of OWL-S), providing a ranking of matches was presented in [12]. In [7, 6] web services discovery through matchmaking was also tackled. An approach for composition was proposed, based on the Difference operator in DLs [35], followed by a concept covering operation optimized using hypergraph techniques. In [32] the location of web services is based on the capabilities they provide. The matching is based only on the inputs and outputs of a web service without taking into account the overall description of the service, which lacks at all in this approach.

In [19] a prototype was developed guiding the user in the dynamic composition of web service. The service filtering is based first on the profile descriptions, with reference to the *ServiceProfile* of OWL-S, then on the constraints that the user specifies at each step of the workflow. The main limits of this approach are primarily two: a human intervention is needed at each step for composing the workflow and the matching between two services is performed taking into account only the output of the first service that could be fed to the second service as input. In [23] a PSM-based approach is proposed for services modeling, where the functional features of a service are described as tasks, while the internal structure of the services are modeled as the methods that solve those tasks. Here only functional information is taken into account for composition. No descriptive information is provided neither used. The same is in the following related works. In [33] they focus on OWL-S ontology to describe the capabilities of web services. The semantic of OWL-S specification is translated to a first-order logic language to allow reasoning on the features of the same services. Encoding service description in a Petri Net formalism provides decision procedures for web service simulation, verification and composition. Petri Nets were proposed both for modeling and orchestration of services in [29, 30], too. In [8] an approach based on finite state machine was presented for automatic synthesis of e-service composition. In [20] also incomplete specification of the client action sequences was taken into account.

WSMF (Web Service Modeling Framework) is proposed in [14] with the aim to provide a conceptual model for developing and describing services; it consists of four elements: *ontologies*, which represent the terminology, *goal repositories*, that describes the problem solved by web services, *web services descriptions* and *mediators*, that deal with the interoperability problems. Another framework developed to describe and execute semantic web service is IRS-II [26] (Internet Reasoning Service). IRS-II is based on the UPML (Unified Problem Solving Method Development Language) framework, which has four category of components specified by means of an appropriate ontology: *do-*

main models, the domain of an application; *task models*, which specify the input, output, preconditions, goal and task to be solved; *Problem Solving Methods (PSMs)*, relate to reasoning processes applied to solve specific tasks; *bridges*, which provide mappings between the components.

8 Conclusion and Future Work

In this work we proposed tractable greedy algorithms to perform an automatic semantic web services orchestration and we presented motivations and examples for the presented approach.

We showed how a semantic specification of the service, based not only on the Inputs Outputs Preconditions Effects model but also endowed with a semantic description of the provided functionalities can be helpful in the orchestration, particularly in the discovery process. Such an approach is also more user oriented as it makes more feasible to devise a request as description of the service looked for rather than a specification based exclusively on invocation information.

We are investigating the extension to negotiable and strict constraints of the precondition specifications and request [17] and how to model the composite web service, computed by *serviceComposer*, with respect to a BPEL4WS specification. A Datalog-based system is also under development able to handle rules for Inputs/Outputs/Preconditions/Effects specifications compliant with SWRL rule language.

References

[1] G. Alonso, et al.. *Web services: Concepts, Architectures and Applications*. Springer Verlag, 2003.

[2] F. Baader, et al. *The Description Logic Handbook*. Cambridge University Press, 2002.

[3] F. Baader, et al. Description logics as ontology languages for the semantic web. *Festschrift in honor of Jörg Siekmann*. Springer-Verlag, 2003.

[4] S. Bechhofer, et al. The dig description logic interface. In *Proc. of DL'03*, 2003.

[5] B. Benatallah, et al. On automating Web services discovery. *VLDB Journal*, 2004.

[6] B. Benatallah, et al. Request Rewriting-Based Web Service Discovery. In *Proc. of ISWC'03*, 2003.

[7] B. Benatallah, et al. Semantic Reasoning for Web Services Discovery. In *Proc. of Workshop on E-Services and the Semantic Web at WWW-03*, 2003.

[8] D. Berardi, et al. Automatic composition of e-services that export their behavior. In *In Proc. of ICSOC-03*, 2003.

[9] T. Berners-Lee, et al. The semantic web. *Scientific American*, 248(4), 2001.

[10] L. Cabral, et al.. Approaches to Semantic Web Services: An Overview and Comparisons. In *Proc. of ESWS '04*, 2004.

[11] A. Calì, et al. Data integration under integrity constraints. *Information Systems*, 29(2):147–163, 2004.

[12] S. Colucci et al. Logic Based Approach to web services discovery and matchmaking. In *Proc. of ICEC'03*, 2003.

[13] S. Colucci, et al. An agency for semantic-based automatic discovery of webservices. In *Proc. of IFIPWCC (AIAI)*, 2004.

[14] D. Fensel and C. Bussler The web service modeling framework WSMF. In *Proc. of the NFS-EU Workshop on Database Inf. Sys. Research for Semantic Web and Enterprises, 1520*, 2002.

[15] T. Di Noia et al. Extending and Computing the Concept Covering for the Semantic Web. Technical report, Tech-Rep n. 21/04/S, http://www-ictserv.poliba.it/PDF/TECH-REP-21-04-2.pdf, 2004.

[16] T. Di Noia et al. Abductive matchmaking using description logics. In *Proc. of IJCAI'03*, 2003.

[17] T. Di Noia et al. Extending Semantic-Based Matchmaking via Concept Abduction and Contraction. In *Proc. of EKAW 2004*. 2004.

[18] T. Di Noia et al. A system for principled Matchmaking in an electronic marketplace. *Intl. Journal of Electronic Commerce*, 8(4):9–37, 2004.

[19] E. Sirin , Hendler J., and Parsia B. Semi automatic composition of web services using semantic descriptions. In *Proc. of the ICEIS Workshop on Web Services: Modeling, Archit. and Infrastructure*, 2003.

[20] D. Berardi et al. Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In *In Proc. of ICSOC-04.*, 2004.

[21] F.Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proc. of IJCAI'03*, pages 319–324, 2003.

[22] M. Frauenfelder. A Smarter Web. *MIT Technology Rev.*, 104(9):52–58, 2001.

[23] A. Gomez-Perez and R. Gonzalez-Cabero. A framework for design and composition of semantic web services. In *Semantic Web Services - 04 AAAI*, 2004.

[24] M-S. Hacid, et al. Computing concept covers: a preliminary report. In *Proc. of DL'02*, 2002.

[25] J.Korhonen ,L. Pajunen , and J.Puustjarvi Automatic Composition of Web Service Workflows Using a Semantic Agent. In *Proc. of IEEE/WIC*, 2003.

[26] E.Motta et al. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Prof. of ISWC '03*, 2003.

[27] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. of VLDB '01*, pages 49–58, 2001.

[28] D.L. McGuinness, et al. DAML+OIL: An Ontology Language for the Semantic Web . *IEEE Intelligent Systems*, 17(5):72–80, 2002.

[29] M. Mecella, F. Parisi Presicce, and B. Pernici. Modeling e-Service Orchestration Through Petri Nets. In *Proc. of VLDB-TES 02*, 2002.

[30] M. Mecella and B. Pernici. Building Flexible and Cooperative Applications Based on e-Services. Technical report, 2002.

[31] Laukkanen Mikko and Helin Heikki. Composing Workflows of Semantic Web Services. In *Proc. of AAMAS* , 2003.

[32] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proc. of ISWC '02*, 2002.

[33] Narayanan S. and McIlraith S. Simulation, verification and automated composition of web services. In *Proc of WWW-02*, 2002.

[34] K. Sycara, et al. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1, December 2003.

[35] G. Teege. Making the difference: A subtraction operation for description logics. In *Proc. of KR'94*, 1994.