Automated Semantic Web Services Orchestration via Concept Covering

T. Di Noia[†], E. Di Sciascio[†], F. M. Donini[‡], A. Ragone[†], S. Colucci[†§]
[†]Politecnico di Bari - via Re David, 200 - I-70125, Bari, Italy
[‡]Università della Tuscia - via San Carlo, 32 - I-01100, Viterbo, Italy
[§]The Open University - Knowledge Media Institute - MK7 6AA, Milton Keynes, U.K.
{t.dinoia,disciascio,a.ragone,s.colucci}@poliba.it;donini@unitus.it

ABSTRACT

We exploit the recently proposed Concept Abduction inference service in Description Logics to solve Concept Covering problems. We propose a framework and polynomial greedy algorithm for semantic based automated Web service orchestration, fully compliant with Semantic Web technologies. We show the proposed approach is able to deal with not exact solutions, computing an approximate orchestration with respect to an agent request modeled a subset of OWL-DL.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]:[retrieval models,search process, selection process];H.4.0 [**Information Systems Applications**]:General;H.3.5 [**Online Information Services**]:[web-based services]

General Terms: Algorithms.

Keywords: semantic web services, orchestration, semantic web, description logics.

1. INTRODUCTION

Web service composition amounts to the orchestration of a certain number of existing web services to provide a composite service that satisfies the user's requirements, in case a single web service is not adequate.

Semantic Web services (SWS) are services endowed of descriptions expressed in a language that has well-defined, and possibly rich, semantics [5]. The aim of such unambiguously machine interpretable descriptions is to ease SWS discovery on the one hand, and orchestration of composite SWS on the other hand, when the requested task cannot be adequately carried out by a single service. Here we 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`". In this paper we propose a framework and an algorithm, fully compliant with the Semantic Web vision and its related technologies, for an automated discovery and composition in a semantic web-services orchestration scenario. To this aim we exploit the recently proposed Concept Abduction [2] inference service in Description Logic to generalize and extend Concept Covering[3] for a subset of OWL-DL.

WWW 2005, May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

2. CONCEPT COVERING VIA CONCEPT ABDUCTION

In [2] the Concept Abduction Problem (CAP) was introduced and defined as a non standard inference problem for Description Logics, 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 $\mathcal{T} \models C \sqcap H \sqsubseteq D$.*

The solution to a CAP can be interpreted as *which part of D is not covered by C*. On the basis of the latter remark in the following we exploit concept abduction to perform a "concept covering". We propose an extension to the basic definition of Concept Covering Problem, eliminating limitations on the DL employed, 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), $\mathcal{V}$=$\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$, 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 $\mathcal{V}$.*

In [3] also the greedy algorithm *GREEDYsolveCCoP* is proposed, in order to compute a solution for a Concept Covering Problem.

3. SEMANTIC WEB SERVICE ORCHESTRATION

The execution of a web service requires its preconditions 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 [4], 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.

3.1 Precondition and Effects for Web Service Orchestration

In order to deal with the execution information, we 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.* **Web Service** [1] : *a triple* $\langle WS_D, P, E\rangle$*, where* WS_D *represents provided service description,* P *the preconditions and* E *the effects.* Hereafter we model WS_D and D as DLs concepts w.r.t. a domain/task ontology $\mathcal{T}_D$. For the sake of simplicity, here P_0,P and E are modeled as conjunction of atomic concepts represented in a Precondition/Effect ontology $\mathcal{T}_{P/E}$. This is a simple TBox with no role and containing only inclusion axioms. A web services composition based exclusively on the solution of $GREEDYsolveCCoP(\mathcal{R}, WS_D, \mathcal{T}_D)$ [3], *i.e.*, solving only a Concept Covering Problem on web services description, cannot deal with the P, E specifications of the services. To this aim we introduce a definition of 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* $i > 1$, $P_0 \sqcap E_1 \sqcap E_2 \sqcap ... \sqcap E_{i-1} \sqsubseteq P_i$.
3. *for* ws_i*, with* $i > 1$*, for each concept name* A *occurring in* E_i, $P_0 \sqcap E_1 \sqcap E_2 \sqcap ... \sqcap E_{i-1} \not\sqsubseteq A$.

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 it possible to 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. an ontology* $\mathcal{T}$*, and* P_0*,* P_i *and* E_i *modeled using a Horn clauses based language. 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.

3.2 Computing a Composite Web Service

We now adapt the basic algorithm proposed in [3] for Concept Covering, to cope with web service preconditions and effects in order to automatically compute a composite web service.For such purpose we need to define an *executable web service* and an *executable set.*

[1]A web service implemented by combining other web services is referred to 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 or composite is transparent to the client, in the following we address them as *web service* for simplicity.

Definition 5. *Given a web service flow* $\mathcal{WSF}(P_0) = (ws_1, ..., ws_n)$*, 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. $\mathcal{WSF}'(P_0) = (ws_1, ..., ws_n, ws^{ex})$ *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 $\mathcal{WSF}(P_0)$.

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* $ws_i \in \mathcal{R}$ *s.t.* 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.

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**
$H = solveCAP(\langle \mathcal{L}, WS_{Di}, D_{uncovered}, \mathcal{T}\rangle)$;
if $H \prec H_{min}$ **then**
$WS_{Dmin} = WS_{Di}$;
$H_{min} = H$;
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

The algorithm returns $\mathcal{CWS}(\langle D, P_0, \mathcal{R}\rangle)$, the composite web service and the uncovered part, $D_{uncovered}$, of the request description D.

4. REFERENCES

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services: Concepts, Architectures and Applications*. Springer Verlag, 2003.

[2] T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proceedings of IJCAI 2003*, pages 337–342, 2003.

[3] T. Di Noia, E. Di Sciascio, and F.M. Donini. 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.

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

[5] Sycara Katia, Paolucci Massimo, Ankolekar Anupriya, and Naveen Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1, December 2003.