A Semantic-based Registry enabling Discovery, Composition and Substitution of Pervasive Services

Michele Ruta
SisInfLab–Politecnico di Bari
via Re David 200
I-70125 Bari, Italy
m.ruta@poliba.it

Tommaso Di Noia
SisInfLab–Politecnico di Bari
via Re David 200
I-70125 Bari, Italy
t.dinoia@poliba.it

Eugenio Di Sciascio
SisInfLab–Politecnico di Bari
via Re David 200
I-70125 Bari, Italy
disciascio@poliba.it

Massimo Paolucci
DoCoMo Communications
Laboratories Europe GmbH
Landsberger Strasse 308-312
80687 Munich, Germany
paolucci@docomolab-euro.com

Floriano Scioscia
SisInfLab–Politecnico di Bari
via Re David 200
I-70125 Bari, Italy
f.scioscia@poliba.it

Eufemia Tinelli
SisInfLab–Politecnico di Bari
Università degli Studi di Bari
via E. Orabona 4
I-70125 Bari, Italy
e.tinelli@poliba.it

ABSTRACT

In this paper we present a semantic-enhanced registry specifically devised for pervasive environments, able to cope with automated mobile service discovery and composition, compliant with OWL-S and with Semantic Web technologies. The proposed approach also deals with non-exact matches (computing an approximate result) and implements a dynamic substitution of services, especially useful in highly unpredictable contexts. It has been implemented and tested in a home/office automation case study, and we also report on experimental results.

1. INTRODUCTION

Novel and powerful mobile devices call for the possibility to adopt advanced discovery and composition of services in ad-hoc environments. Though increasingly effective, such devices have specific limitations, which have to be taken into account when designing systems able to support mobile users. We propose an innovative framework for mobile service discovery, composition and substitution. Knowledge Representation techniques and approaches are exploited and adapted to highly flexible and volatile ubiquitous computing frameworks. In particular, we propose an extended version of the open source *jUDDI*[1] implementation by the Apache Software Foundation. Building on previous works that enhanced the basic discovery features of it with semantic-based capabilities [16], we devise an adaptation as well as an extension of that system to pervasive scenarios. The registry adopts an OWL-S 1.1 Profile instance[2] annotation of mobile services. Ontology-based metadata are exploited in order to perform a semantic-based discovery and composition w.r.t. a given request. We also implement a basic technique for dynamic substitution of failed or no longer available services.

W.r.t. general purpose semantic-based service composers, the implemented approach goes beyond a discovery of subsumption relations between preconditions and effects (or inputs and outputs) of a service components sequence, being also able to cope with: (1) non-exact matches and (2) effects duplication.

(1) If it is not possible to compute a sequence of mobile service components satisfying the whole user request, an approximate solution is given. An explanation of what is still uncovered by the computed composite service is also provided, as complementary result of the composition.

(2) In order to execute a mobile service component belonging to a complex flow, its preconditions (inputs) must be satisfied, possibly using information provided by other services. Moreover care has to be paid in avoiding the duplication of effects (outputs) when composing services.

To perform discovery and composition of mobile services, we exploit structured queries over a database. Recall that the relational model implies –due to its intrinsic structure– the possibility to establish well known relationships among generic entities. Hence it can be correctly exploited to elicit new information starting from the one stated within a specific model instance. The proposed framework has been targeted to pervasive environments. It exploits an m-DBMS, *i.e.*, Oracle 10g Lite [13], and it basically consists of a mobile registry able to cope with semantically annotated OWL-S resource discovery and composition.

The remaining of the paper is organized as follows. Section 2 outlines motivation of the paper. Section 3 presents fundamentals of both the language we adopted and allowed reasoning tasks. The proposed framework is reported in Section 4 after a brief summary on related work. The case study in Section 5 clarifies the approach and the rationale behind

[1] http://ws.apache.org/juddi/

[2] OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/

it and Section 6 reports a sketch of the proposed architecture. It describes the prototype implementing the approach and outlines obtained experimental results. Section 7 closes the paper.

2. MOTIVATION

The framework we devised is an evolution of common service discovery architectures (Figure 1). It aims to enable a really mobile approach to discovery and composition of pervasive services avoiding the employment of computationally-demanding reasoners.

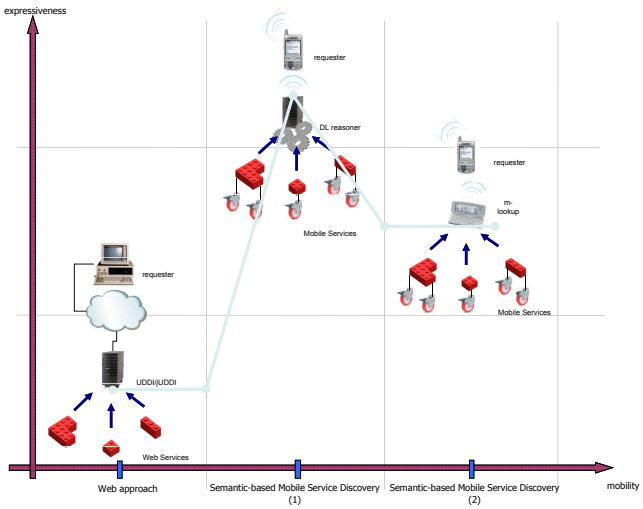

Figure 1: The service discovery evolution process

A rough analysis of computational costs of semantic-based discovery and composition points out the expressiveness of adopted formalisms plays a critical role. As explained hereafter, we selected a sublanguage deriving from OWL DL[3] to model ontologies, requests and service annotations. It is obvious that w.r.t. applications which involve fixed reasoners (in our previous implementations we adopted *MAMAS-tng*[4] matchmaker), the proposed approach suffers from the reduced expressiveness of the managed logic. This is the main price to be paid to the increased mobility level. Nevertheless, a comparison with common directory services on the Web shows that the proposed system offers a better quality of the provided discovery/composition and it allows the substitutability feature which is of undoubted interest in volatile contexts.

Our framework is easily adaptable to a widespread class of applications where a large number of low complexity components can be aggregated to build composed services with growing semantic relevance. The model we propose is surely suitable for environments disseminated by many resources having just a low level of semantic sophistication (*e.g.*, either data coming from cluster head sensors or RFID object annotations [18] and so on). In general, pervasive contexts are characterized by a large amount of small services/resources rather than heavy and complex ones. This is obviously due to the computational and power supply limitations of mobile providers and to their short storage availability. Hence, it can be more appropriate to shape "agile" service discovery architectures able to assemble and orchestrate *on the fly* a lot of elementary components rather than complex and heavy structures somewhat uncontrollable.

It has to be noticed that the semantic complexity of the overall system is always controlled. A simple modeling of the proposed service composition architecture points out that the "physical" distribution of the system –in terms of number of service instances– typically increases when decreases the semantic consistence of services components involved in the discovery. On the contrary, a reduced number of service components is necessary when they have a greater semantic depth. This trivial consideration can justify the idea that service discovery frameworks as the one we propose here maintain a limited complexity. It particularly corresponds to the complexity level (depth) of the reference ontology model. The lesser is the semantic complexity of the chosen TBox, the higher will be the pervasiveness of the resulting architecture.

3. BACKGROUND FRAMEWORK

In this section, we introduce some preliminary notions related to the language we adopted and reasoning tasks.

The Language. In order to reduce computational costs due to the full expressiveness of OWL DL, we selected a subset of OWL DL where ***class descriptions*** (also used to represent mobile service descriptions and queries) consist of:

- class identifier (a URI reference);
- `owl:complementOf` class identifier;
- datatype property restriction;
- the intersection of two or more class descriptions.

Furthermore, we use ontologies where all the axioms contain a class identifier only in their left hand side and for `rdfs:subClassOf` and `owl:equivalentClass` axioms, only one axiom is allowed for each class identifier. For `owl:disjointWith` axioms, both left hand side and right hand side are class identifiers such that they do not appear as left hand side of `owl:equivalentClass` axioms[5]. Possible axioms are then in the form:

- *class identifier* `rdfs:subClassOf` *class description*
- *class identifier* `owl:equivalentClass` *class description*
- *class identifier* `owl:disjointWith` *class identifier*

Due to the structure of such ontologies and class descriptions, classical ontology reasoning tasks such as subsumption and consistency check can be easily reduced to set comparison, drastically decreasing the computational needs. Given two class descriptions –*e.g.*, a mobile service description MSD and a query Q– and an ontology $\mathcal{T}$ (for Terminology) proceed as follows:

For each class identifier C in MSD and Q:
  if the axiom C `rdfs:subClassOf` D is in $\mathcal{T}$
    recursively rewrite C as `owl:intersectionOf` $\{C, D\}$;
  if the axiom C `owl:equivalentClass` D is in $\mathcal{T}$

[3]OWL Web Ontology Language, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/
[4]http://sisinfab.poliba.it/MAMAS-tng/
[5]Using Description Logics [1] terminology, these ontologies are known as *simple-TBox* [9].

recursively replace C with D;
if the axiom C `owl:disjointWith` D is in $\mathcal{T}$
recursively rewrite C as `owl:intersectionOf` $\{C, \texttt{owl:complementOf}\ D\}$;

After this preprocessing step, known as *unfolding* [1], both MSD and Q are rewritten as a conjunction of class identifiers, negated class identifiers, datatype property restrictions.

Subsumption. In order to verify if MSD is **subsumed by** (is more specific than) Q, just check if for each conjunct C_i in Q, C_i is also a conjunct of MSD.

Disjointness. If one wants to check if Q and MSD are **disjoint with each other** it suffices to check if there exists a class identifier C_i in Q such that `owl:complementOf` C_i is a conjunct of MSD (the same is if we flip over Q and MSD). In other words we consider the conjunction of elements in Q and MSD and, in general, in every class expression allowed by the language we chose, as sets of elements.

Concept Abduction. Based on this set-based formalization, also solutions to a **concept abduction problem** (CAP) [8] can be easily computed. Roughly, a CAP can be described as: given two class descriptions MSD and Q, such that MSD is not subsumed by Q –*i.e.*, the mobile service description does not completely satisfy the query– hypothesize a class expression H representing what is underspecified in MSD in order to be more specific than (subsumed by) Q. From an operational point of view, for each conjunct C_i in Q, check if C_i is also a conjunct of MSD. If not, hypothesize C_i and add it to H. We write $H = solveCAP(MSD, Q)$ to indicate that H is what has to be hypothesized and added to MSD in order to completely satisfy the request Q. Actually, minimality criteria on the size of H have to be defined and adopted in order to avoid trivial and redundant solutions. In particular, for class expression represented as a conjunction of elements in [8] *irreducible solutions* are defined as solutions that are minimal w.r.t. the number of conjuncts (taking into account also the axioms in the ontology $\mathcal{T}$).

Concept Covering. The definition of concept abduction is also useful to define another non-standard reasoning task: **concept covering**. Informally, given a set of class descriptions $\mathcal{R} = \{MSD_i\}$, find the subset $\mathcal{R}' = \{MSD'_i\} \subseteq \mathcal{R}$ such that the solution H is minimal (given a minimality criterion) for CAP computed w.r.t. `owl:intersectionOf`$\{MSD'_i\}$ and Q. We say $\mathcal{R}'$ **covers**, or **is a covering for**, Q. In this case H represent the uncoverd part of Q. In case $H = \emptyset$, we say to have a **full covering**.

4. MOBILE SERVICE DISCOVERY, COMPOSITION AND SUBSTITUTION

In mobile service oriented architectures, a requester generally ignores the underlying ad-hoc network structure as well as available services. Hence discovery protocols must be able to identify both the consistence and the location of services. Nevertheless current paradigms are usually based on a true/false matching of attributes, which is surely limited for advanced applications [7]. In fact, due to the connectivity features made available by modern handheld devices, users require appropriate performances [4]. In recent years, dynamic distributed approaches have been devised. For example service composition in ubiquitous environments is the issue discussed in [3], where a protocol based on mobile broker agents is presented. The composition of mobile services starts with the election of a broker node which locates mobile resources. On the contrary, Ponnekanti et al. in [15] present an application to solve the issue of interoperability among services in ubiquitous computing. The authors aim to show how applications can interact with services even when globally unique interfaces are not provided. In [10] the use of a rule engine to dynamically determine a near-optimal provider for each requested service is implemented. This multi-provider platform reduces dependencies; it increases geographic and functional coverage and it allows load balancing. [11] introduces a framework for resource retrieval based on a set of self-organized discovery agents which manage a directory service where resources can be searched out by using a hash indexing. In addition, the proposed system enables a dynamic selection of the best service provider according to supplied QoS.

In general, we can say semantics allows to overcome limitations of traditional discovery protocols, but several issues have to be solved in order to adapt both ideas and technologies devised for the WWW to unpredictable environments like mobile ad-hoc ones. The backward-compatible semantic-based Bluetooth Service Discovery Protocol (SDP) proposed in [17] allowed the management of both syntactic and semantic discovery of services, by integrating a semantic layer within the protocol stack at application level. Here we present a general framework for a semantic-based automated composition/substitution of mobile services. We basically hypothesize a directory-centric architecture. That is, a restricted set of nodes in the ad-hoc environment act as lookup and orchestration servers and the other ones assume only a passive role.

Let us suppose a generic client submits a request for a service to the registry. Hence the lookup collects various component services coming from nearby nodes. It attempts to cover the request starting the composition algorithm. If retrieved services do not allow to completely fulfill the request, an approximate solution has to be taken into account, possibly explaining the approximation and letting the requester accept it or not. Observe that the role of the directory/composer server, which performs the discovery and the orchestration, is fully interchangeable with any host within the **M**obile **A**d hoc **NET**work (MANET) so that composition is not bound to the computational capabilities of a specific client.

4.1 Service composition

When it comes to orchestration of discovered elementary resource components in order to return as output a complex service to the user, a service composition algorithm has to be adopted. It accepts in input, among other things, preconditions that must be satisfied, returning the composite mobile service best matching user requests.

Following the IOPE (Input, Output, Precondition, Effect) model [14], the discovery of a composite service in the MANET requires to take into account also mobile services execution information, *i.e.*, precondition and effect specifications[6]. In fact some services may set specific requisites to be satisfied and, if a mobile device does not grant them, it cannot use the service.

If the service is a component of a set, its preconditions must be satisfied possibly using information provided by

[6]In this paper, for the sake of conciseness, we do not consider also inputs and outputs.

other component services. Moreover, when composing services, a duplication of effects can happen. Here we adapt the service discovery and composition models in [16] and [5] to deal with a pervasive scenario. In particular we define:

Mobile Service: $ms = \langle P, MSD \rangle$ where MSD is the description of provided service effects and P its preconditions.

User Request: $r = \langle P_0, Q \rangle$ where Q represents requester desired effects and P_0 the initial precondition/information it provides.

Available Preconditions: Given a sequence of mobile service $\mathcal{CMS} = (ms_k = \langle P_k, MSD_k \rangle)$, $k = 1 \dots N$ and a request r, we call *Available Preconditions* at step $N+1$, the class expression built as the conjunctions of P_0 and all the effects MSD_k. Formally, AP_{N+1} = `owl:intersectionOf` $\{P_0, MSD_1 \dots MSD_N\}$.

Executable Service: Given a sequence of mobile service $\mathcal{CMS}$ and a request r, an executable service $ms^{ex} = \langle P^{ex}, MSD^{ex} \rangle$ is a service such that:

- AP_{N+1} is subsumed by P^{ex} — service preconditions are satisfied by available preconditions;
- no conjunct[7] of MSD^{ex} subsumes AP_{N+1} –if a conjunct of MSD^{ex} is implied by AP_{N+1}, this means that the effect it represents is already provided by services in $\mathcal{CMS}$. In this way, the duplication of effects is avoided.

We call **Executable Set** of $\mathcal{CMS}$ the set of all the executable services for $\mathcal{CMS}$. We indicate such set with $\mathcal{EX}(\mathcal{CMS})$. If $\mathcal{CMS} = \emptyset$ then $\mathcal{EX}(\mathcal{CMS})$ is the set of services whose preconditions are satisfied by P_0.

Composite Mobile Service. Given a request $r = \langle P_0, Q \rangle$, a composite mobile service is a sequence of mobile services $\mathcal{CMS}$ $(ms_k = \langle P_k, MSD_k \rangle)$, $k = 1 \dots N$ such that:

- P_0 is subsumed by MSD_1;
- ms_{k+1} is an executable service for ms_k;
- $\mathcal{R}(\mathcal{CMS}) = \{MSD_1 \dots MSD_N\}$ **covers** Q (the effects of all the mobile services in $\mathcal{CMS}$ is a covering for Q).

Given a set $\mathcal{R}$(for Repository) of service descriptions, the *serviceComposer* algorithm in Figure 2 returns a Composite Mobile Service $\mathcal{CMS}$ and, in case $\mathcal{R}$ is not a full covering for Q, the uncovered part H.

Notice that in line 11 a function is used to compare two concepts H and H_{min}. In its basic version, given a class expression C, $d(C)$ computes the number of conjuncts in C after *unfolding*. Actually, such function can be customized to take into account other (contextual) parameters such as QoS, the distance of the service from the requester and so on (see below). In case of pervasive environments, the composition algorithm has to consider, among others, the influence of distance between offered and requested services. In fact the set of component services is not assigned *a priori*, but it may change according to network characteristics. In spite of increasing covering possibilities, the involvement of nodes farther and farther implies a greater risk in the persistence of links among requester and the set of providers. Hence it is useful to define a metric/weight which takes into account distance (in terms of number of hops) from service requester

[7]As defined in Section 3, the sub-language we selected allows only `owl:intersectionOf` class descriptions.

Algorithm $serviceComposer(\mathcal{R}, \langle D, P_0 \rangle, \mathcal{T})$

input a set of services $\mathcal{R} = \{ms_i = \langle P_i, MS_D(i) \rangle\}$, a request $r = \langle P_0, Q \rangle$
output $\langle \mathcal{CMS}, H \rangle$

```
1  begin algorithm
2   CMS = ∅;
3   Q_uncovered = Q;
4   H_min = Q;
5   do
6    compute EX(CMS);
7    MSD_min = ∅;
8    for each ms_i ∈ EX(CMS)
9     if (CMS, ms_i) is a composite mobile service then
10     H = solveCAP(MSD_i, Q_uncovered);
11     if d(H) < d(H_min) then
12      MSD_min = MSD_i;
13      H_min = H;
14     end if
15    end if
16   end for each
17   if MS_Dmin ≠ ∅ then
18    R = R\{ms_i};
19    CMS = ⟨CMS, ms_i⟩;
20    Q_uncovered = H_min;
21   end if
22  while(MSD_min ≠ ∅);
23  return ⟨CMS, Q⟩;
24 end algorithm
```

Figure 2: The algorithm *serviceComposer*.

to service providers for weighing the matching degree. Services that are "located" on mobile devices in proximity of the requester have the maximum weight and this weight progressively reduces with distance. A logarithmic function is adopted. In fact it presents a behavior approximately proportional to the distance for a short number of hops, but it varies very slowly over a greater distance limit.

Hence, in line 11 of the *serviceComposer* we correct $d(H)$ (and $d(H_{min})$) by computing the influence of "physical distance" from the requester. A $(1 + log_{10} n)$ factor –where n is the number of hops from requester to provider– is introduced. Services at one hop of distance, *i.e.*, in the radio range of the requester, are not influenced by this corrective factor. Instead, for services at two or more hops of distance, the *semantic distance* is increased of a logarithmic factor. Nevertheless, if there is a mobile service description full covering Q (or $Q_{uncovered}$), the influence of physical distance is absent. The rationale is that, in the presence of a perfectly matching service, we accept to tolerate the overload and the risk of using far resources.

4.2 Service substitutability

During the discovery phase, all services composing a set should be identified. Nevertheless, in a pervasive environment, we cannot take for granted that all of them will be permanently and simultaneously available. In fact, during the execution, a service could fail or, due to host mobility, it could become unreachable. In these cases, in a dynamic fashion, the system should substitute mobile services no more suitable with better ones.

To implement the substitution capability we define a *Similarity Group* of a mobile service ms as a collection of services which can be substituted with ms [6]. Now the classification of a service to determine if it belongs to the group is the central question. Specific conditions have to be defined to enable the substitutability of a mobile service with another

one [2].

Notice that the *Similarity Group* for a mobile service ms (from now on $\mathcal{SG}(ms)$) is a simple set of services without any seeming order relation. An order relation could be esteblished based on the proximity among resources, but due to the host mobility it is inapplicable to $\mathcal{SG}(ms)$. Also notice that $\mathcal{SG}(ms)$ is created w.r.t. each service in $\mathcal{CMS}$ so each constituent service can have a set of substitutes.

Given ms, to build $\mathcal{SG}(ms)$, information about candidate substitutes is required prior to admit them in the substitutability class [12]. Furthermore we need some data about the interface of a service, *i.e.*, required preconditions and provided effects; this is essential for evaluating its correct insertion in a mobile service flow $\mathcal{MSF}$. In order to decide if a generic service can belong to a *Similarity Group* conditions about preconditions and effects have to be verified. Let us suppose $(ms_1, \ldots, ms_i, \ldots, ms_N)$ are services in $\mathcal{CMS}$ and let us imagine we want to constitute the similarity group $\mathcal{SG}(ms_i)$. We say $ms_j^{sub} \in \mathcal{SG}(ms_i)$ iff the following conditions hold:

1. ms_j^{sub} is an *executable mobile service* for $(ms_1, ms_2, ..., ms_{i-1})$
2. for $h = i+1 ... N$, ms_h is an *executable mobile service* for $(ms_1, ms_2, ..., ms_{h-1})$

This is the theoretical framework but, in practice, the substitutability is implemented in a more quick way. In fact, given $\mathcal{CMS}$ and ms_i to be substituted, in order to verify if an executable service ms_j^{sub} is suitable, we take the available preconditions AP_i and we recalculate AP_k (with $k = i+1 \ldots N$) checking its satisfiability. In other words, if AP_i is the available information for ms_i and if we want to substitute it, the new $AP'_{i+1} = \texttt{owl:intersectionOf}\{AP_i, MSD_j^{sub}\}$ has to be determined. Eventually, AP'_{i+1} will be used to check the suitability of the next ms_{i+1} and so on. If one of these checks fails, we can conclude that the ms_j^{sub} is unsuitable. In the progressive substitution of services in $\mathcal{CMS}$ it is desirable to start with first services in the flow, to reuse the already determined AP in the next substitution steps and consequently reduce the overhead deriving from this processing. Hence, if $ms_i \in \mathcal{CMS}$ suddenly turns unavailable, we can take a service in $\mathcal{SG}(ms_i)$ and substitute it.

The *Similarity Group* of service ms_i is created at discovery phase, but it is more and more enriched while the discovery progresses when new services are discovered extending the search to the next hop. Hence, if $\mathcal{SG}^k(ms_i)$ is the *Similarity Group* of the service ms_i at hop k, we can say $\mathcal{SG}(ms_i) = \mathcal{SG}^1(ms_i) \cup \mathcal{SG}^2(ms_i) \cup \ldots$.

5. CASE STUDY

The proposed approach for semantic-based service composition has been evaluated in a case study for a simulated smart control application. The scenario encompasses a home automation platform and it also follows the user in further environments –car, office– according to pervasive computing paradigms. Controllers manage individual appliances and devices within a smart environment, such as lighting and alarms. Bluetooth wireless connectivity allows easier installation and direct user control from her Bluetooth-enabled PDA/phone.

In such a system, three service layers can be identified, with increasing semantic complexity.

1. Elementary service layer (EL): each elementary service allows to control directly the operating parameters of a single managed resource, *e.g.*, intensity of a light source or temperature of air conditioning. The whole service platform comprises a large number of such services. They typically have very simple preconditions, or even none.

2. Automation service layer (AS): provides capabilities comparable to current home/office automation solutions. Services in this layer have a richer semantics, involving multiple devices of the same kind simultaneously, in order to adapt the environment to preset use cases, *e.g.*, "maximum comfort for relax" and "power-saving mode". Hence, a service in this layer is defined in terms of a number of elementary services, requiring a composition process.

3. Lifestyle service layer (LS): provides services, defined completely from the user's perspective. It allows to adapt the environment to user preferences and current activities, expressed with high-level semantics. Manifold subsystems are involved and coordinated at this level, therefore different services from the AS layer have to be composed. Preconditions may consist in state of equipment and/or context information. The latter can be collected from external sources.

An ontology was developed to model properties of the three service layers in our home automation case study. Due to lack of space, Figure 3 reports only some axioms that are particularly useful to understand the following examples (classical logic notation is adopted hereafter for easier reading).

The LS service layer exploits semantically rich concepts, which can be tied to external information sources, such as the calendar in the user's PDA. Keywords in calendar entries are exploited to recognize user location (*e.g.*, home or workplace) and activity (work, sport, relax) throughout the day. Such knowledge is used along with date/time and weather information in order to adapt the behavior of the smart control facility to maximize user comfort, energy efficiency and security.

Let us consider a small example. *Rose lives in a smart home consisting in a living room, kitchen, bedroom and bathroom. Every morning, her PDA checks her daily calendar and plans behavior of the home automation system. Today Rose is going to be at work from 8.30 to 12.30.* A requester-centric and Bluetooth-based semantic service composition allows to coordinate the different devices in the home environment.

The first composite service to be required maps the user activity of leaving home for several hours. In that case, energy efficiency and home security are the primary concerns. Available preconditions include current time and weather information collected by the PDA. Let us see how this problem is modeled.

$r_1 = \langle P_{r_1}, Q \rangle = \langle \; Morning \sqcap = 0 \; Occupants \sqcap Warm\_Weather, Leaving\_House \rangle$

As explained in Section 3, when $\texttt{r}_1$ is issued, the system unfolds it according to the axioms in the domain ontology. Unfolding is recursive: the requested LS service is expressed in terms of AS services, which are in turn rewritten in terms of EL services:

$r_1 = \langle = 0 \; Occupants \sqcap Morning \sqcap Warm\_Weather, Home\_Heating\_Off \sqcap Bedroom\_Light\_Off \sqcap Living\_Room\_Light\_Off \sqcap Kitchen\_Light\_Off \sqcap Bathroom\_Light\_Off \sqcap TV\_Off \sqcap PC\_Off \sqcap Oven\_Off \sqcap All\_Windows\_Shut \sqcap Security\_Sensors\_On \sqcap$

Leaving_House ≡ Power_Saving_Mode ⊓ Security_System_On ⊓ Redirect_Communications
Power_Saving_Mode ≡ Home_Heating_Off ⊓ All_Lights_Off ⊓ Optional_Appliances_Off
All_Lights_Off ≡ Bedroom_Light_Off ⊓ Living_Room_Light_Off ⊓ Kitchen_Light_Off ⊓ Bathroom_Light_Off
Optional_Appliances_Off ≡ TV_Off ⊓ PC_Off ⊓ Oven_Off
Security_System_On ≡ Security_Sensors_On ⊓ Security_Camera_On ⊓ Door_Lock_Closed ⊓ All_Windows_Shut
Redirect_Communications ≡ Telephone_Routing_On ⊓ Entry_Phone_Routing_On
Car_Comfort ≡ Car_Radio_On ⊓ Car_Radio_Program_Fit ⊓ Car_Air_Conditioning_On ⊓ Car_Air_Comfort

Figure 3: Axioms in the home automation ontology used in the case study

$Security\_Camera\_On \sqcap Door\_Lock\_Closed \sqcap Telephone\_Routing\_On \sqcap Entry\_Phone\_Routing\_On\rangle$

The *serviceComposer* algorithm can now be applied. Let us suppose that the following EL services are available in the home environment:

$ms_1 = \langle P_1, MSD_1\rangle = \langle$TV_On, DVD_Movie$\rangle$
$ms_2 = \langle$Music_On, Classical_Music$\rangle$
$ms_3 = \langle\top,$Living_Room_Lights_Soft$\rangle$
$ms_4 = \langle=$ 0 Occupants, Home_Heating_Off$\rangle$
$ms_5 = \langle$Cold_Weather, Home_Heating_High$\rangle$
$ms_6 = \langle$Mild_Weather, Home_Heating_Low$\rangle$
$ms_7 = \langle$Warm_Weather, Home_Heating_Off$\rangle$
$ms_8 = \langle=$ 0 Occupants ⊓ Door_Lock_Closed, Security_Sensors_On$\rangle$
$ms_9 = \langle=$ 0 Occupants ⊓ Door_Lock_Closed, Security_Camera_On$\rangle$
$ms_{10} = \langle\top,$Door_Lock_Closed$\rangle$
$ms_{11} = \langle\top,$All_Windows_Shut$\rangle$
$ms_{12} = \langle\top,$Bedroom_Light_Off$\rangle$
$ms_{13} = \langle\top,$Living_Room_Light_Off$\rangle$
$ms_{14} = \langle\top,$Kitchen_Light_Off$\rangle$
$ms_{15} = \langle\top,$Bathroom_Light_Off$\rangle$
$ms_{16} = \langle\top,$TV_Off$\rangle$
$ms_{17} = \langle\top,$TV_On$\rangle$
$ms_{18} = \langle\top,$Music_On$\rangle$
$ms_{19} = \langle\top,$PC_Off$\rangle$
$ms_{20} = \langle\top,$Oven_Off$\rangle$
$ms_{21} = \langle\top,$Telephone_Routing_Off$\rangle$
$ms_{22} = \langle\top,$Telephone_Routing_On$\rangle$
$ms_{23} = \langle\top,$Entry_Phone_Routing_Off$\rangle$
$ms_{24} = \langle\top,$Entry_Phone_Routing_On$\rangle$

It can be noticed that several elementary services do not need preconditions. Other ones, *e.g.*, heating, depend on external parameters, therefore multiple service instances are provided to represent all the possible operating modes of a device. The selection of the correct operating mode is based on available preconditions.

1a. The composition algorithm starts with:
$\mathcal{CMS} = \emptyset$
$Q_{uncovered} = Home\_Heating\_Off \sqcap Bedroom\_Light\_Off \sqcap Living\_Room\_Light\_Off \sqcap Kitchen\_Light\_Off \sqcap Bathroom\_Light\_Off \sqcap TV\_Off \sqcap PC\_Off \sqcap Oven\_Off \sqcap All\_Windows\_Shut \sqcap Security\_Sensors\_On \sqcap Security\_Camera\_On \sqcap Door\_Lock\_Closed \sqcap Telephone\_Routing\_On \sqcap Entry\_Phone\_Routing\_On$

1b. After the first step, the following results are obtained:
$\mathcal{EX}(\mathcal{CMS}) = \{ms_3, ms_4, ms_7, ms_{10}, ms_{11}, ms_{12}, ms_{13}, ms_{14}, ms_{15}, ms_{16}, ms_{17}, ms_{18}, ms_{19}, ms_{20}, ms_{21}, ms_{22}, ms_{23}, ms_{24}\}$
$\mathcal{CMS} = (ms_4, ms_{10}, ms_{11}, ms_{12}, ms_{13}, ms_{14}, ms_{15}, ms_{16}, ms_{19}, ms_{20}, ms_{22}, ms_{24})$
$Q_{uncovered} = Security\_Sensors\_On \sqcap Security\_Camera\_On$

Notice that ms_4 and ms_7 are both executable mobile services and they have the same effect. The composition algorithm adds only the first one to $\mathcal{CMS}$, so that unneeded effect duplication is avoided.

1c. The second step of the algorithm then produces:
$\mathcal{EX}(\mathcal{CMS}) = \{ms_3, ms_7, ms_8, ms_9, ms_{20}, ms_{21}\}$
$\mathcal{CMS} = (ms_4, ms_{10}, ms_{11}, ms_{12}, ms_{13}, ms_{14}, ms_{15}, ms_{16}, ms_{19}, ms_{20}, ms_{22}, ms_{24}, ms_8, ms_9)$
$Q_{uncovered} = \top$

A full covering has been reached and the composition now stops. The result of the composition is stored in the PDA. *When Rose is actually leaving her house, she activates the composite service from her PDA with a single command.* Using Bluetooth communications, the PDA enacts mobile services in the same order as they appear in $\mathcal{CMS}$.

When Rose enters her car, her PDA adapts the environment to her preferences. A second mobile service composition takes place in the user's car. She does not need an explicit interaction with the system, since her habits and preferences are already modeled, adopting the reference ontology. Her PDA issues the following request to service providers within the car:

$r_2 = \langle Morning \sqcap Warm\_Weather, Car\_Comfort\rangle = \langle Morning \sqcap Warm\_Weather, Car\_Radio\_On \sqcap Car\_Radio\_Program \sqcap Car\_Air\_Conditioning\_On \sqcap Car\_Air\_Conditioning\_Comfort\rangle$

Let us suppose that the following EL services are available in the car environment:

$ms_{25} = \langle\top,$Car_Radio_Off$\rangle$
$ms_{26} = \langle\top,$Car_Radio_On$\rangle$
$ms_{27} = \langle$Car_Radio_On ⊓ Afternoon, Car_Radio_Music_Program ⊓ Car_Radio_Program_Fit$\rangle$
$ms_{28} = \langle\top,$Car_Air_Conditioning_Off$\rangle$
$ms_{29} = \langle\top,$Car_Air_Conditioning_On$\rangle$
$ms_{30} = \langle$Car_Air_Conditioning_On ⊓ Warm_Weather, Car_Air_Conditioning_Fresh ⊓ Car_Air_Comfort$\rangle$
$ms_{31} = \langle$Car_Air_Conditioning_On ⊓ Cold_Weather, Car_Air_Conditioning_Warm ⊓ Car_Air_Comfort$\rangle$

2a. The *serviceComposer* algorithm starts with:
$\mathcal{CMS} = \emptyset$
$Q_{uncovered} = Car\_Radio\_On \sqcap Car\_Radio\_Program\_Fit \sqcap Car\_Air\_Conditioning\_On \sqcap Car\_Air\_Comfort$

2b. The first step products the following results:
$\mathcal{EX}(\mathcal{CMS}) = \{ms_{25}, ms_{26}, ms_{28}, ms_{29}\}$
$\mathcal{CMS} = (ms_{26}, ms_{29})$
$Q_{uncovered} = Car\_Radio\_Program\_Fit \sqcap Car\_Air\_Comfort$

2c. After the second step, the following results are obtained:
$\mathcal{EX}(\mathcal{CMS}) = \{ms_{30}, ms_{31}\}$
$\mathcal{CMS} = (ms_{26}, ms_{29}, ms_{30})$
$Q_{uncovered} = Car\_Radio\_Program\_Fit$

2d. The outcome of the third step is:
$\mathcal{EX}(\mathcal{CMS}) = \emptyset$
$\mathcal{CMS} = (ms_{26}, ms_{29}, ms_{30})$
$Q_{uncovered} = Car\_Radio\_Program\_Fit$

Composition terminates with incomplete covering of the request. In particular, for the sake of the example we have supposed that a suitable radio program could not be found (preconditions for executing ms_{27} are not met). When full request satisfaction cannot be achieved, the service composer reports the outcome. The underlying semantics of service descriptions helps give a comprehensible explanation

of what has not been found.

Rose enters her office. The environment is automatically adapted to her preferences and calendar, as well as to external conditions. With a similar process as in the first use case, the PDA performs a service composition process to adapt every environmental condition to user preferences: alarms are switched off, office climate and lighting are adjusted, user's workstation is powered on and scheduled appointments are synchronized. Due to the lack of space, this use case is not reported here.

Finally, it is useful to remark that the proposed approach –based on Knowledge Representation– allows a mobile service platform to evolve without central administration. Using an ad-hoc wireless network and a common vocabulary to describe services, new mobile service providers can join the system without any human effort for hardware installation or software integration. Discovery and composition capabilities of the outlined system are fully dynamic, hence new services can be immediately used.

6. SYSTEM IMPLEMENTATION AND EX-

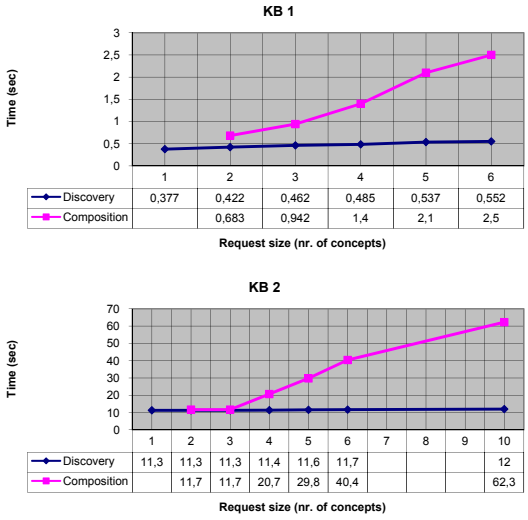

Figure 4: Time progression of discovery and composition process w.r.t. request size

The implementation allowed to validate the approach, test algorithms behavior, and carry out experiments. We model each service specification as an OWL-S 1.1 Profile instance. The main component of the system is a module performing the discovery of services via an m-RDBMS and computing a ranked list taking into account incomplete or missing information. In what follows we briefly describe the proposed E/R model featuring each component table and we show related experimental results.

- `Parents_0` table is built after an analysis of the OWL ontology. It contains all the first degree "parent/child" relationships also expressed by means of possible properties.
- `Parents_i` tables (i=1..N) are built expanding all the relationships of order higher than one among concepts. `Parents_i` is derived joining `Parents_i-1` and `Parents_0`.
- `Ancestors` table is given by joining all the `Parent_i` (i=0..N) and resumes all the subsumption relationships among concepts provided with the selected ontology.
- `Resources` table collects service descriptions. Each tuple will contain a component concept with a possible role.
- `Normalized` table is obtained joining `Resources` and `Ancestors` tables. It will contain all the relationships among service instances and related parents.

Due to the sublanguage we use to model ontologies, service descriptions and queries, the proposed model allows to perform the discovery procedure. The information stored within the database is used to compute on the fly the "unfolded" version of Q and of mobile service descriptions. These unfolded class descriptions are then used to solve corresponding concept abduction problems –needed by *serviceComposer*– as described in Section 3. The discovery procedure receives in input the set of conjuncts in Q. They are individually considered and, for each of them, all the parents within the `Ancestors` table are extracted. The corresponding query is:

```
SELECT parent
FROM Ancestors
WHERE child = <component concept>
```

This collection of parents will be then used for selecting services from `Normalized` table containing, in their semantic annotation, at least a class among them within the just created set. The corresponding query is:

```
SELECT service
FROM Normalized
WHERE class IN (<parent list>)
```

The proposed approach has been implemented and tested on an HP iPAQ 2210h PDA (all tests have been performed using a fully charged device battery). Two different ontologies (not reported for brevity) have been used. The first one (Onto1) contains approximately 50 among classes and properties. The second one (Onto2) contains approximately 100 among classes and properties. The number of service instances is 6 for Onto1 and 33 for Onto2. In Figure 4 average time progression of discovery and composition procedures w.r.t. size of request (in terms of conjuncts number) is reported.

A general comparison of bootstrap phase duration against the composition one (computed in the worst case, *i.e.*, attempting a composition over a 10 concepts request exploiting Onto2) points out that the initial processing is prevailing, as revealed by Figure 5. Time consumption for mapping the Knowledge Base into the DB is relevant and in particular the creation of `Parents_i` and `Normalized` tables takes up most of bootstrap time. Due to this important limitation, mapping operations are performed only after ontology agreement and only if necessary. That is the KB mapping happens only the first time a mobile directory has to "parse" a specified KB. Until the registry is still in the same application context, it will preserve the previously mapped KB.

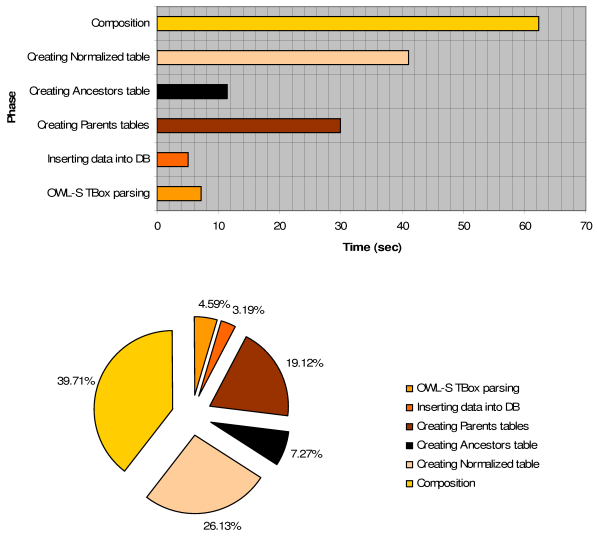

Figure 5: Time consumption general comparison

7. CONCLUSION AND FUTURE WORK

We have presented a semantic-based registry specifically aimed at pervasive environments, and using a mobile-oriented DBMS. It is able to cope with automated discovery, composition and substitution of mobile services compliant with OWL-S and with Semantic Web technologies. The approach has been implemented in a prototype and some preliminary experimental results have been reported.

Future research aims to refine the proposed architecture with optimizations able to enhance the expressiveness of allowed logic formalisms also coping with both restricted storage availability and computational capabilities of mobile devices. We are also evaluating the exploitation of an m-ODBMS to increase performances and to extend permitted reasoning tasks.

Acknowledgments

Some of the authors acknowledge partial support of Apulia Region Strategic Project PS_121.

8. REFERENCES

[1] F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.

[2] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In *TES*, Lecture Notes in Computer Science, pages 15–28. Springer, 2004.

[3] D. Chakraborty, Y. Yesha, and A. Joshi. A Distributed Service Composition Protocol for Pervasive Environments. In *Wireless Communcations and Networking Conference (WCNC)*. IEEE, March 2004.

[4] H. Chen, A. Joshi, and T. Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents MeetJini in the Aether. *Cluster Computing*, 4(4):343–354, 2001.

[5] F. Colasuonno, S. Coppi, A. Ragone, L. Scorcia, T. Di Noia, and E. Di Sciascio. jUDDI+: A Semantic Web Services Registry enabling Semantic Discovery and Composition. In *The 8th IEEE Conference on E-Commerce Technology and the 3rd IEEE Conference on Enterprise Computing*, pages 442Ű–444.

[6] V. De Antonellis, M. Melchiori, B. Pernici, and P. Plebani. A Methodology for e-Service Substitutability in a Virtual District Environment. In *CAiSE '03*, LNCS, pages 552–567. Springer, June 2003.

[7] T. Di Noia, E. Di Sciascio, and F. Donini. Semantic matchmaking as non-monotonic reasoning: A description logic approach. *Journal of Artificial Intelligence Research (JAIR)*, 29:269–307, 2007.

[8] T. Di Noia, E. Di Sciascio, F. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 337–342, 2003.

[9] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in Description Logics. In G. Brewka, editor, *Principles of Knowledge Representation: Studies in Logic, Language and Information*, pages 191–236. CSLI Publications, 1996.

[10] M. Hörhammer and P. Biswas. A Rule Engine for Location Based Content Syndication. In *MDM '04*, pages 88–93. IEEE, January 2004.

[11] J. Liu, Q. Zhang, B. Li, W. Zhu, and J. Zhang. A Unified Framework for Resource Discovery and QoS-Aware Provider Selectionin Ad Hoc Networks. *ACM Mobile Computing and Communications Review*, 6(1):13–21, January 2002.

[12] M. Mecella, B. Pernici, and P. Craca. Compatibility of e-services in a Cooperative Multi-platform Environment. In *TES*, Lecture Notes in Computer Science, pages 44–57. Springer, 2001.

[13] Oracle Corporation. *Oracle Database Lite 10g R2 - Feature Overview*, June 2006.

[14] OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/.

[15] S. Ponnekanti and A. Fox. Application-Service Interoperation without Standardized Service Interfaces. In *PerCom '03*, pages 30–37. IEEE, March 2003.

[16] A. Ragone, T. Di Noia, E. Di Sciascio, F. Donini, S. Colucci, and F. Colasuonno. Fully Automated Web Services Discovery and Composition through Concept Covering and Concept Abduction. *International Journal of Web Services Research (JWSR)*, 4(3):85–112, 2007.

[17] M. Ruta, T. Di Noia, E. Di Sciascio, and F. Donini. Semantic-Enhanced Bluetooth Discovery Protocol for M-Commerce Applications. *International Journal of Web and Grid Services*, 2(4):424–452, 2006.

[18] M. Ruta, T. Di Noia, E. Di Sciascio, F. Scioscia, and G. Piscitelli. If objects could talk: A novel resource discovery approach for pervasive environments. *International Journal of Internet and Protocol Technology (IJIPT)*, 2(3/4):199–217, 2007.