m-jUDDI+: a Semantic-enabled Service Registry for Discovery, Composition and Substitution in Pervasive Environments

Michele Ruta[1], Tommaso Di Noia[1], Eugenio Di Sciascio[1], and Francesco Maria Donini[2]

[1] Politecnico di Bari, via Re David 200, I-70125 Bari, Italy,
[m.ruta, t.dinoia, disciascio]@poliba.it
[2] Università della Tuscia, via San Carlo 32, I-01100 Viterbo, Italy,
donini@unitus.it

Abstract. We present a semantic-based version of UDDI registry, specifically devised for pervasive environments, able to cope with automated mobile service discovery and composition and compliant with Semantic Web technologies. The framework exploits non-standard inference services within OWL-S. The approach also deals with effect duplication and non-exact matches, computing an approximate covering result. A simple technique is also presented to implement a dynamic substitution of services/resources failed or no longer available, especially useful in highly unpredictable frameworks. The proposed approach has been implemented and tested in an ubiquitous computing environment.

1 Introduction

This paper presents a novel framework for service discovery, composition and substitution in a Mobile Ad-hoc NETwork (MANET) environment. Knowledge Representation techniques and approaches are exploited and adapted to highly flexible and volatile ubiquitous computing frameworks. In particular, we introduce m-jUDDI+, an extended version of the open source *jUDDI* [1] implementation by the Apache Software Foundation. m-jUDDI+ is the mobile version of jUDDI+ [2], and it adopts an OWL-S 1.1 Profile instance [3] annotation of mobile services. Ontology-based metadata are exploited in order to perform a semantic-based discovery of a single service as well as a semantic-based composition w.r.t. a given request. We also implement a dynamic substitution of failed or no longer available services.

W.r.t. general purpose semantic-based service composers, the approach implemented in m-jUDDI+ goes beyond the "simple" discovery of subsumption relations between preconditions and effects (or inputs and outputs) of a service components sequence, but it is 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, the best approximate solution is given based on

the request. An explanation of what is still uncovered by the computed composite service is also provided, as complementary result of the composition process.

(2) In order to execute a mobile service, its preconditions (inputs) must be satisfied, possibly using information provided by other services within the flow. Moreover care has to be paid in avoiding the duplication of effects (outputs) when composing services, which might be also due to entailment relationships among different effects provided by services being composed (see later on for further details). After the execution of the composite service we do not want two or more services providing the same features, even partially overlapping.

The proposed approach has been targeted to pervasive environments exploiting an m-DBMS, *i.e.*, Oracle 10g Lite [4] to implement a mobile registry able to cope with semantically annotated OWL-S resource discovery and composition. In what follows both implementation details as well as experimental results are presented.

The remaining of the paper is organized as follows. Section 2 presents a greedy algorithm based for automated mobile service composition. The substitutability approach is described in Section 2.2. Section 3 reports a sketch of the proposed architecture whereas Section 4 describes the prototype implementing the approach, its integration in the OWL-S 1.1 framework, and outlines obtained experimental results. Section 5 closes the paper.

2 Mobile Service Discovery, Composition and Substitution

An ad-hoc network is a system in rapid evolution, providing services and/or exposing resources. A generic requester may generally ignore the network structure as well as available services. Hence discovery protocols allow to identify consistence and location of services. Nevertheless current service discovery paradigms are usually based on syntactic matching of attributes, which are largely inadequate for advanced mobile applications. Users need to submit articulate requests and to receive appropriate replies [5]. Semantics allows to overcome these limitations, 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 SDP proposed in [6] allows the management of both syntactic and semantic discovery of services and resources, by integrating a semantic layer within the protocol stack at application level. Unused classes of 128 bit service identifiers in the original Bluetooth (UUIDs) are exploited to label specific ontologies, naming this value OUUID (Ontology Universally Unique IDentifier). By means of the OUUID matching between request and available resources the context can be determined and a preliminary selection of resources referring to the same ontology of the request performed. A general description of each resource of the environment can then be stored within a mobile device as a database record labeled with a unique 32-bit handle. Each record entirely consists of an OUUID, a human-readable name for the resource, a resource description expressed in OWL-S syntax and

a variable number of utility attributes (*i.e.*, numerical values used according to specific applications). By adding four Service Discovery Protocol PDUs to the original standard (exploiting not used PDU IDs), further semantic-enabled discovery functionalities have been introduced, without limiting the original SDP capabilities. The overall interaction is based on the original application layer in Bluetooth. No modifications were made to the original structure of transactions.

When it comes to orchestration of discovered elementary resource components in order to give in output a complex service to the user, *serviceComposer* algorithm plays a central role. It accepts in input, among other things, preconditions that must be satisfied, providing in output the composite mobile service best matching the user requests.

2.1 Composition of Mobile Services

Here we present the general framework for a semantic-based automated composition of mobile services. Let us suppose a generic client submits a request for a service in an ad-hoc network. We basically hypothesize a requester-centric composition system. The requester searches for a complex service and collects various component services coming from nearby nodes. Hence, it attempts to cover the request starting a composition algorithm. In the proposed approach, the requester acts as the service orchestrator and other nodes in the ad-hoc environment assume only a passive role.

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 assumed by the requester, which performs the composition, is fully interchangeable with any host within the MANET so that composition is fully decentralized. Various resource providers may take part to it. Each host concurs to cover the whole request or part of it, by means of service descriptions it manages.

To explain and motivate the approach and the rationale behind it, we present a simple semantic based service discovery model and we add to it service composition features, enriching the model. Hereafter, we adopt Description Logic (DL) formalism [7], assuming the reader being familiar with it. In the initial architecture we define both the request D and the description of each available service on a mobile device, as DL concept descriptions w.r.t. an ontology $\mathcal{T}$ shared among some users in the network. We perform a preliminary selection procedure based on ontology numeric identifiers to extract devices that manage the same ontology $\mathcal{T}$. Hence, given a request D modeled w.r.t. $\mathcal{T}$, in what follows we briefly outline the service discovery algorithm computed by the service requester to retrieve resources possibly satisfying her request:

1. Extract service descriptions in the requester cache.
2. Select all the resource descriptions which refer to the same $\mathcal{T}$.
3. Put all the retrieved descriptions in a set $\mathcal{R}$.
4. Call resource composition algorithm with input $\mathcal{R}$, D and $\mathcal{T}$.
5. Is there an exact solution?

(a) If yes, the algorithm outputs the set of services representing the exact solution to the composition. Exit.
(b) If not, the algorithm outputs the set of services representing an approximate solution to the composition and an explanation on why the solution is not an exact one. Exit.

The composition of discovered services in the MANET requires to take into account also their execution information, *i.e.*, inputs, outputs, precondition and effect specifications. In fact some services may set specific execution requisites to be satisfied. Here if a mobile device does not manage them, it cannot use the service. 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. Now we briefly explain precondition and effects function in the orchestration of mobile services and we show how to compose services. In order to use a service on a mobile device, its preconditions must be satisfied and, if the service is a component of a service set, this has to be done possibly using information provided by other component services. Moreover, when composing services, a duplication of effects can take place. Here we extend the service composition model in [2] to deal with a pervasive scenario. For the sake of clarity we only recall main terms and definitions. In particular we define:

Mobile Service: a triple $\langle MS_D, P, E\rangle$ where MS_D is the description of provided service, P its preconditions and E the effects. Furthermore, indicating with AI_i the available information for the i-th mobile service ms_i and with E_j the effects produced by ms_j, with $j < i$, the following relation ensues: $AI_i = P_0 \sqcap E_1 \sqcap E_2 \sqcap ... \sqcap E_{i-1}$.

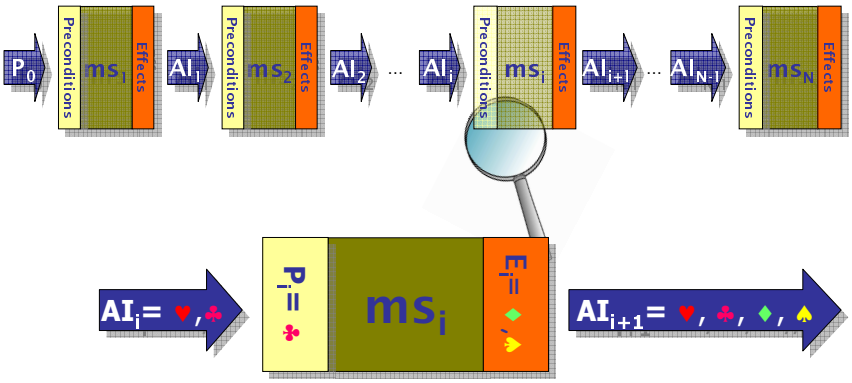

Fig. 1. A composition framework scheme

Based on the definition of mobile service flow w.r.t. some initial preconditions P_0 –from now on $\mathcal{MSF}(P_0)$ [2]– here we define a **composite mobile service** w.r.t. a request D. A ***composite mobile service*** for $\langle D, P_0\rangle$ w.r.t. the set of discovered mobile services $\mathcal{R}$, from now on $\mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)$, is a mobile

service flow such that for each ms_j in the execution flow: $D_{\mathcal{CMS}(\langle D,P_0,\mathcal{R}\rangle)} = \{MS_D(j)|ms_j \in \mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)\}$ covers D.

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

The *serviceComposer* algorithm outputs the resulting $\mathcal{CMS}$ as well as $D_{uncovered}$. If a requester, with the evolution of the MANET, receives descriptions of new services belonging to other far service providers in the network, it can run again the composition algorithm and can try to better cover the request.

Notice that in a mobile scenario the discovery and composition of services is performed by an association of nodes which take part to the final result. Due to the volatility of such a context, the set of discovered services available for the further orchestration is variable and in evolution. Hence, the composing algorithm should be run in subsequent and coordinate sessions. w.r.t. Web services composition on wired systems [2], in this case we have more subsequent composition processes. The orchestration goes through several steps, producing a progressive refinement of results.

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

input a set of services $\mathcal{R} = \{ms_i = \langle MS_D(i), P_i, E_i\rangle\}$, a request $\langle D, P_0\rangle$ - where D and $MS_D(i)$ are satisfiable in $\mathcal{T}$-
output $\langle \mathcal{CMS}, H\rangle$

```
1  begin algorithm
2   CMS(⟨D, P0, R⟩) = ∅;
3   D_uncovered = D;
4   H_min = D;
5   do
6    compute EX_{CMS(⟨D,P0,R⟩)};
7    MS_Dmin = ⊤;
8    for each ms_i ∈ EX_{CMS(⟨D,P0,R⟩)}
9     if D_{CMS(⟨D,P0,R⟩)} ∪ {MS_D(i)} covers D_uncovered then
10     H = solveCAP(⟨L, MS_D(i), D_uncovered, T⟩);
11     if H ≺ H_min then
12      MS_Dmin = MS_D(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(⟨D, P0, R⟩) = (CMS(⟨D, P0, R⟩), ms_i);
20    D_uncovered = H_min;
21   end if
22  while(MS_Dmin ≢ ⊤);
23  return ⟨CMS(⟨D, P0, R⟩), D_uncovered⟩;
24 end algorithm
```

Furthermore, in case of pervasive environments, the composition algorithm has to take into account the influence of distance between offered and requested service. In fact the set of components 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 in the MANET implies a greater risk in the persistence of links among requester and set of providers.

Hence it is useful to define a metric which takes into account distance (in terms of number of hops) from service requester to service providers for weighting the matching degree. Services which are "located" on mobile devices in proximity of requester have the maximum weight and this weight progressively reduces with distance. A logarithmic function is adopted. In fact it presents a growth approximately proportional to the distance for a short number of hops, but assumes values almost constant over a specified limit.

In line 11 of the algorithm we correct the "semantic distance" from offered service to request 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 requester, have a *semantic distance* determined by their real semantic similarity w.r.t. the request. Instead, for services at two or more hops of distance, the *semantic distance* is increased of a logarithmic factor. Notice that distances over 10 hops substantially double the *semantic distance*. Nevertheless, if there is an exact match the influence of physical distance is absent. This means that, in the presence of a possible exactly matching service, we accept to tolerate the overload and the risk of using far resources.

2.2 Service Substitutability

During the discovery phase, all services composing a set should be identified. Nevertheless, in a pervasive environment, rarely all of them can be assumed to be simultaneously available. In fact, during the execution, a service could fail and, dut to host mobility, it could become unreachable. Furthermore, since a MANET is an extremely evanescent system, while the composition is in progress, a better resource could be detected as well as newer releases of an already discovered service could be made available. 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* as a collection of component services which can be substituted with each other [8]. Now the classification of a servce to determine its belonging to the group is the central question. A set of rules for substitution of a mobile service with another one has to be defined [9].

Notice that the *Similarity Group* (from now on $\mathcal{SG}$) is a simple set of services without any order relation. This is strictly correlated to the variable structure of a MANET. So it is unnecessary to order the class of substitutable services because they are too instable. The only reasonable order relation should be based on the proximity among resources, but due to the host mobility it is inapplicable to the $\mathcal{SG}$. Also notice that a $\mathcal{SG}$ is created w.r.t. each service in the $\mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)$ so each constituent service can have a set of substitutes.

To build the $\mathcal{SG}$, information about candidate substitutes are required prior to admit them in the substitutability class [10]. 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 the $\mathcal{MSF}(P_0)$. In order to decide if a generic service can belong to a *Similarity Group* two conditions about

preconditions and effects have to be verified. Let us suppose $(ms_1, ms_2, ..., ms_N)$ are services in a $\mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)$ and let us imagine we want to constitute the ms_i similarity group $\mathcal{SG}$(i). We say $ms_j^{sub}(j = 1...L) \in \mathcal{SG}$(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 compact way. In fact, to verify if a substitute service ms_j^{sub} is suitable, we take into account the already available AI_i (if ms_i is the service to substitute) and we recalculate next AI_k (with $k = i + 1...N$) checking the satisfiability of the respective ms_k. In other words if AI_i is the available information for ms_i and if we want to substitute the same ms_i, the new $AI'_{i+1} = AI_i \sqcap E_j^{sub}$ has to be determined. Hence it will be used for checking the satisfiability of the next ms_{i+1} and so on. If one of these checks fails, we can conclude that the ms_j^{sub} is effectively unsuitable. In the progressive substitution of services in a $\mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)$ it is desirable to start with first services in the flow, to reuse the already determined AI in the next substitution steps and consequently reduce the overhead deriving from this processing.

Hence, if the generic ms_i belonging to $\mathcal{CMS}(\langle D, P_0, \mathcal{R}\rangle)$ suddenly turns unavailable, we can take a generic ms_j^{sub} from the $\mathcal{SG}$(i) with $j = 1...L$ and substitute it.

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

In addition to the fault tolerance feature, the substitutability can also be used as load balancing of services in case of high services concentration on a single device or group of devices.

3 Basic Architecture

The testbed prototype we implemented is an evolution of traditional service discovery architectures (Figure 2). It enables a fully decentralized approach to discovery and composition of mobile services.

`m-jUDDI+`, plays a fundamental role within the whole architecture. It copes with OWL-S based annotation of mobile services. 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. In order to deal with the MS_D we extended the Result class in the OWL-S 1.1 Process by means of a new $< owl : ObjectPropertyrdf : ID = "effectsDescription" / >$. The property has OWL-S *Result* class as domain and $< owl : Thing / >$ as range; the cardinality is restricted to 1, because (in this initial prototype) its specification refers to a description w.r.t. a task ontology identified by a unique

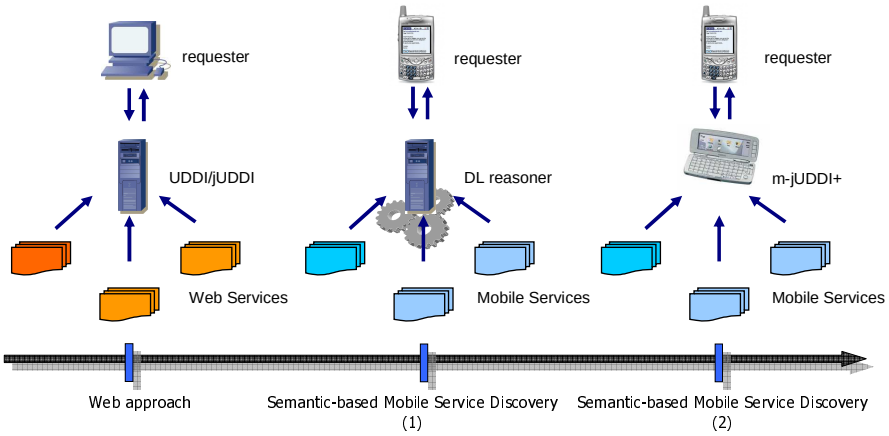

Fig. 2. The service discovery evolution process

OUUID identifier. In Figure 3 modifications to the standard OWL-S 1.1 definition are reported.

In what follows we sketch the structure of the proposed architecture and briefly outline a typical interaction. Main component of `m-jUDDI+` is the **Service Selector** module (*SS*). It performs the discovery of services, exploiting an m-DBMS and a set of relational tables and computes a ranked list taking into account incomplete or missing information.

Input to the system is the $\mathcal{CMS}(\langle P_0, E, \mathcal{R}\rangle)$ where P_0 are the initial preconditions/inputs, E the desired effects/outputs after service(s) execution (*i.e.*, the user requests) whereas $\mathcal{R}$ set is filled by services referred to the selected ontology. In the mobile, OWL-S compliant `m-jUDDI+`, the composition process is performed through the following steps:

1. User and composer (from now on *hotspot*) agree about the reference ontology by means of an ontology matching procedure performed via the semantic-enhanced Bluetooth Service Discovery Protocol (SDP) [6].
2. Exploiting the selected ontology, the user formulates a request in terms of required preconditions and provided effects, using its OWL Classes and Properties.
3. User composes her request, which is then transmitted to the *hotspot* via the semantic-enhanced Bluetooth SDP [6], together with the initial preconditions P_0 she is able to provide.
4. The *hotspot* selects service instances referred to the agreed ontology and sends them to the selector *SS*, together with the ontology itself.
5. The semantically annotated OWL description of the request E and P_0, are sent to the *SS*.
6. The overall information is mapped to the appropriate set of relational tables.

```
<profile:serviceName>
     Service Name
</profile:serviceName>
<profile:textDescription>
     A human-understandable description of the service
</profile:textDescription>
<profile:serviceCategory>
     <addParam:OUUID rdf:ID="OUUID-category">
          <profile:value>
               OUUID value
          </profile:value>
          <profile:code>
               OUUID code
          </profile:code>
     </addParam:OUUID>
</profile:serviceCategory>
<profile:hasPrecondition>
     A logic expression referring to concepts in the  T_P/E ontology
</profile:hasPrecondition>
<profile:hasResult>
     <process:Result rdf:ID="HaveSeatResult">
          <process:hasEffect>
               A logic expression referring to concepts in the  T_P/E ontology
          </process:hasEffect>
          <process:effectDescription>
               An OWL-DL expression referring to the concepts in the domain ontology
                         identified by the OUUID
          </process:effectDescription>
     </process:Result>
</profile:hasResult>
```

Fig. 3. An OWL-S mobile service Profile instance

7. The *hotspot* with the *SS* provide the $\mathcal{CMS}$ also computing a matching degree and, if an approximation occurs, an explanation about the uncovered part of the request is given. Resulting information are returned to the requester.

SS component computes the service best matching a request w.r.t. the selected $\mathcal{T}$ so preparing components for the further composition phase. For each ms, a check has to be performed in order to verify the compatibility between user provided preconditions/inputs and service preconditions/inputs. In case, the MS_D is selected as a candidate to satisfy part of the request E. For each candidate a check between provided effects/outputs and required ones has to be done. Each compatible service/resource is ranked in order to evaluate its degree of correspondence w.r.t. the request.

The composition process implies several calls to the *SS* module, *i.e.*, the orchestration is performed in more coordinate and subsequent sessions. In the first one the selector attempts to identify the component service which better covers the whole request. In case of failure, during each ensuing session it provides the service which better deals with the uncovered part of the request. Each found service is progressively added to the $\mathcal{CMS}$ and the uncovered section is updated for the following step. Such an iteration goes on until the request is fully covered or compatible component services are totally exploited[3]. What is still unfulfilled,

[3] Notice that a service can be defined "compatible" with the request if it is composed by at least one concept also present in the request annotation itself. Not compatible

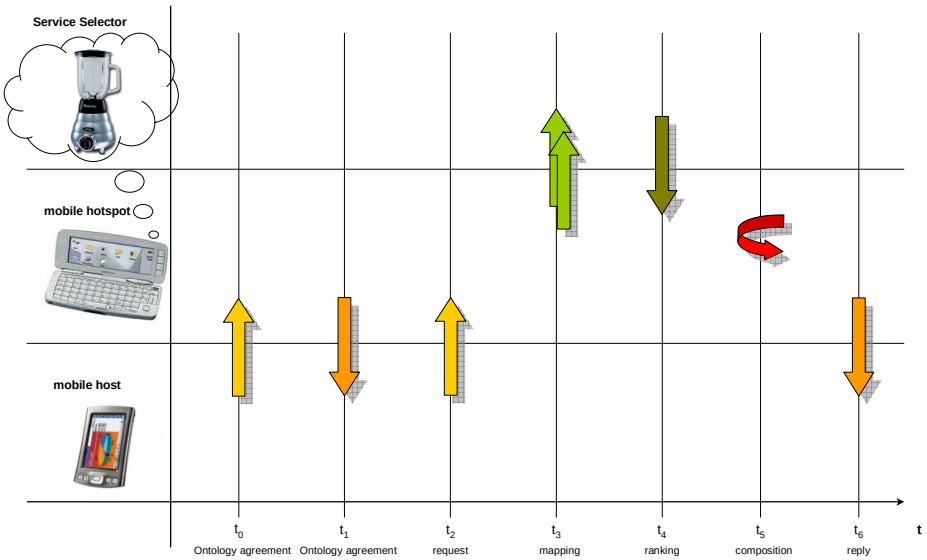

Fig. 4. Service composition framework typical interaction

in case, will compose the rest H: an explanation about the approximate solution to be provided to the user.

4 System Implementation and Experiments

A generic XML schema of an OWL-S ontology only presents subsumption relations which can be mapped into a relational database collection. The proposed approach aims to avoid employing in pervasive frameworks computationally-demanding reasoners. To perform discovery and composition of mobile services, their functionalities are substituted –to some extent– with structured queries over the DB set. 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 ones stated within a specific model instance. In what follows we briefly describe the proposed E/R model featuring each component table and show related experimental results.

– `Parents_0` table is built after a parsing process involving the XML file of the OWL-S ontology. It contains all the first degree "parent/child" relationships also expressed by means of possible roles.
– `Parents_i` table (i=1..N) are built expanding all the relationships of order higher than first among concepts. `Parents_i` is derived joining `Parents_i-1` and `Parents_0`.

services are immediately discarded and they are not inserted within the composition flow $\mathcal{MSF}(P_0)$.

- `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 services descriptions. Each tuple will contain a component concept with a possible role.
- `Normalized` table is obtained joining `Resources` table and `Ancestors` one. It will contain all the relationships among service instances and related parents.

The proposed model is able to cope with an elementary discovery procedure which is the basic feature of *Service Selector* component described above. The adopted algorithm is outlined hereafter. Discovery procedure receives in input the list of concepts of the request contained in a hash-set. They are considered individually and for each of them all the parents within the `Ancestors` table are extracted. The correspondent query is:

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

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

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

Retrieved services will be successively compared with the ones in the $\mathcal{MSF}$ and discarded if already inserted. In case, for each candidate component service a rank value will be computed indicating the degree of correspondence with request. Best matching service is selected and inserted within the $\mathcal{MSF}$. If no instances result suitable for enriching the composition flow, the process is halted. The adopted formula for the rank computation ensues:

$$rank = Retrieved\ Concepts \cdot \left(1 - \frac{Not\ retrieved\ Concepts}{Total\ Concepts}\right)$$

The proposed approach has been implemented and tested over a HP iPAQ 2210h PDA. Two different ontologies (not reported for brevity) have been used. The first one (Onto 1) contains approximately 50 among concepts and roles. The second one (Onto 2) contains approximately 100 among concepts and roles. Correspondent KBs respectively manage 6 and 33 service instances. In what follows we report time consumption for the bootstrap phase of the application in both cases. Then an average time progression of discovery and composition procedures w.r.t. consistence of request (in terms of component concept number) is reported.

A general comparison of bootstrap phase duration against composition one (computed in the worst case, *i.e.*, attempting a composition over a 10 concepts

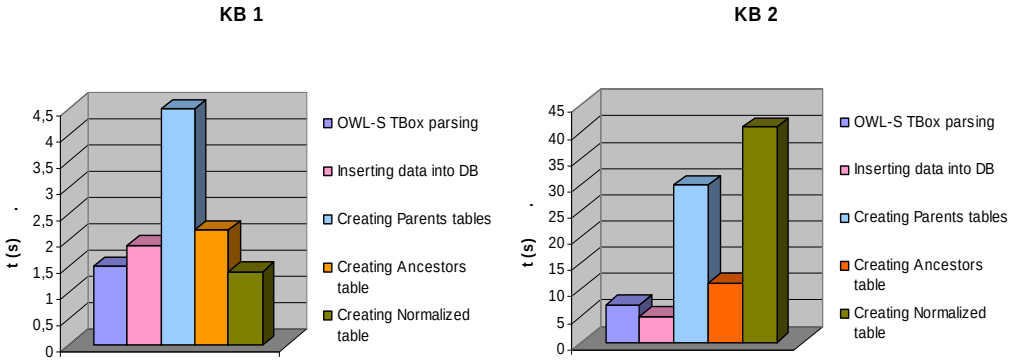

Fig. 5. Bootstrap phase time consumption

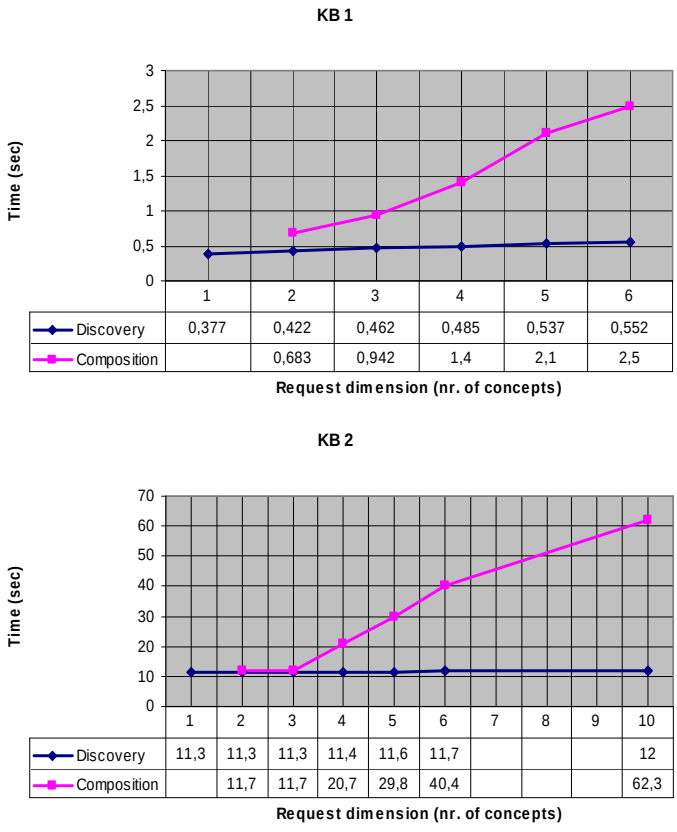

Fig. 6. Time progression of discovery and composition process w.r.t. request dimension

request exploiting Onto 2) points out that the initial processing is prevailing. 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 bootstrap required time. Due to this important limitation, mapping operations are performed only after ontology agreement and KB instantiation and are not executed if not necessary. Anyhow the proposed system showed a better behavior –in terms of response time– w.r.t. a traditional reasoner over a wired server. It has been noticed that response time noteworthily decreases when battery is only partially charged. All the carried out tests have been performed taking into account optimal condition of the wireless device. That is the PDA had a fully charged battery.

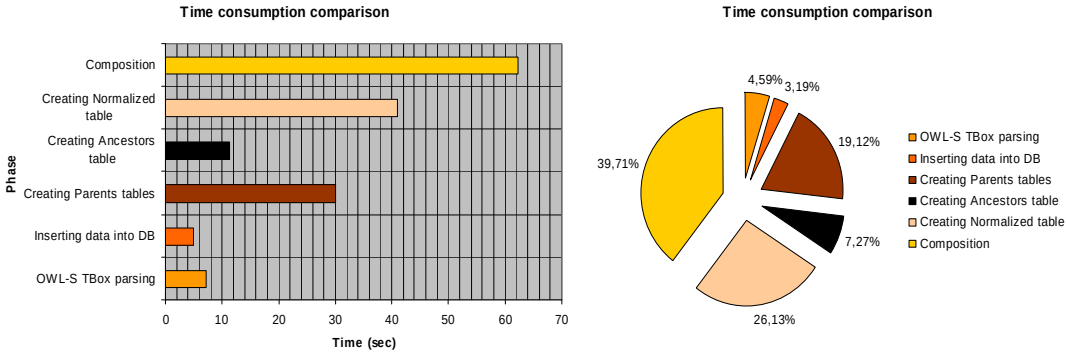

Fig. 7. Time consumption general comparison

5 Conclusion and Future Work

Novel and powerful mobile devices call for the possibility to adopt discovery and composition services in MANET environments. Obviously, though powerful, such devices have specific limitations, which have to be taken into account to provide useful results in a reasonable amount of time and keeping into account computational limits. Addressing these issues, in this paper we presented `m-jUDDI+`, a semantic-based version of UDDI registry specifically devised for pervasive environments, and using a mobile-oriented DBMS, able to cope with automated mobile service discovery and composition, compliant with OWL-S and with Semantic Web technologies. The approach also deals with effect duplication and non-exact matches, computing an approximate result, and implements a dynamic substitution of services, especially useful in highly unpredictable frameworks. The proposed framework has been implemented and tested in an ubiquitous computing environment.

Future research aims to refine the proposed architecture with optimizations able to cope with both restricted storage availability and computational capabili-

ties of mobile devices. Furthermore a complex framework –integrating semantic-based Service Discovery infrastructures for MANETs devised in [11]– will be built and tested in order to allow an integration of decentralized discovery and composition algorithms with semantic-enhanced discovery protocols. The porting of the system under SQLite [12] will enable to embed proposed orchestration features in general purpose pervasive architectures involving technologies as RFID, Bluetooth, 802.11.

Acknowledgments

We wish to acknowledge support of EU FP-6 IST STREP TOWL project "Time-determined ontology based information system for real time stock market analysis".

References

1. Apache Software Foundation: jUDDI. (http://ws.apache.org/juddi/)
2. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F., Colucci, S., Colasuonno, F.: Fully Automated Web Services Discovery and Composition through Concept Covering and Concept Abduction. International Journal of Web Services Research (JWSR) **4**(3) (2007) Scheduled To appear.
3. OWL-S: Semantic Markup for Web Services: (http://www.daml.org/services/owl-s/1.1/overview/)
4. Oracle Corporation: Oracle Database Lite 10g R2 - Feature Overview. (2006)
5. Chen, H., Joshi, A., Finin, T.: Dynamic Service Discovery for Mobile Computing: Intelligent Agents MeetJini in the Aether. Cluster Computing **4**(4) (2001) 343–354
6. Ruta, M., Di Noia, T., Di Sciascio, E., Donini, F.M.: Semantic-enhanced bluetooth discovery protocol for m-commerce applications. International Journal of Web and Grid Services **2**(4) (2006) 424–452
7. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2002)
8. De Antonellis, V., Melchiori, M., Pernici, B., Plebani, P.: A Methodology for e-Service Substitutability in a Virtual District Environment. In: CAiSE '03. LNCS, Springer (2003) 552–567
9. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In: TES. Lecture Notes in Computer Science, Springer (2004) 15–28
10. Mecella, M., Pernici, B., Craca, P.: Compatibility of e-services in a Cooperative Multi-platform Environment. In: TES. Lecture Notes in Computer Science, Springer (2001) 44–57
11. Ruta, M., Di Noia, T., Di Sciascio, E., Donini, F.: Semantic-Enhanced Bluetooth Discovery Protocol for M-Commerce Applications. International Journal of Web and Grid Services **2**(4) (2006) 424–452
12. SQLite: Distinctive features of SQLite. http://www.sqlite.org/different.html (2006)