



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

Michele **Ruta**<sup>1</sup> , Tommaso **Di Noia**<sup>1</sup> , Eugenio **Di Sciascio**<sup>1</sup> , Francesco Maria **Donini**<sup>2</sup>

<sup>1</sup>**Politecnico di Bari, Bari, Italy**

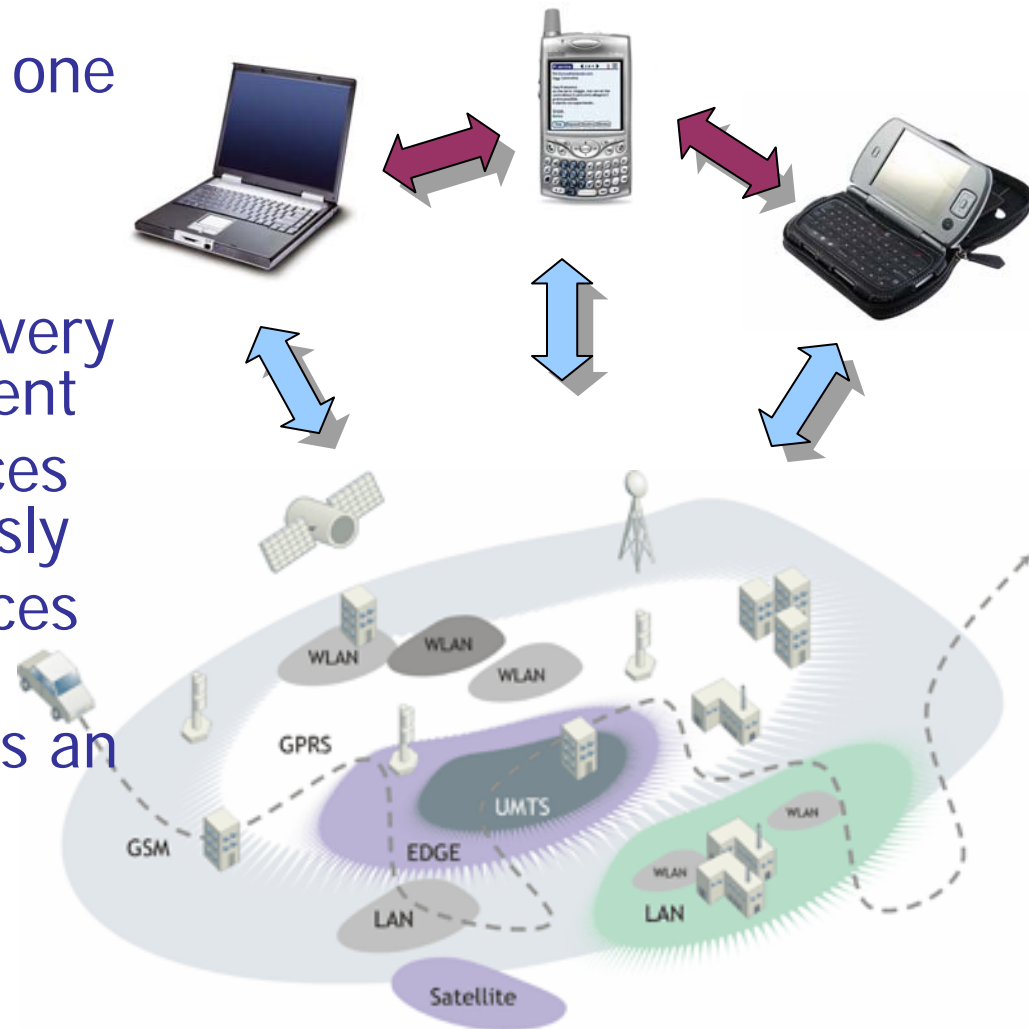
<sup>2</sup>Università della Tuscia, Viterbo, Italy



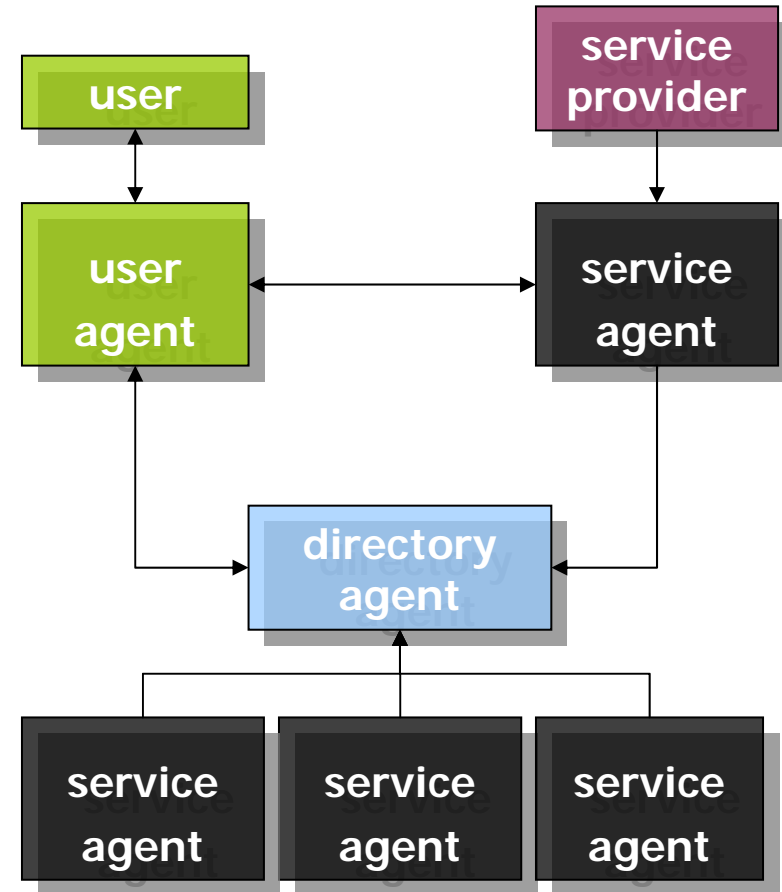
# Outline of the talk

- Service Discovery in Pervasive Environments (issues and perspectives)
- State of the art
  - Lookup services: UDDI/jUDDI
  - Semantic-based Bluetooth SDP
- Semantic based Service Composition
- Service Substitutability
- **m-jUDDI+**: framework and approach
- Performance evaluation
- Conclusion and Future Work

- In mobile environments one or more devices provide and/or use services/resources
- An ad-hoc network is a very unpredictable environment
- Location of mobile devices could change continuously
- Information about services are often unavailable
- Service Discovery (SD) is an essential feature



- Mobile service discovery paradigms have been obtained by **adapting** protocols designed for wired contexts (Jini, Salutation, Service Location Protocol, UPnP or **UDDI**)
- Common service discovery architectures use a **centralized** and **registration-oriented** mechanism
  - Each service advertises and registers itself to a service lookup
  - The directory service keeps track of resources in the network
  - The lookup server attempts to match the query pattern with resource descriptions in its DB
- Current SDPs require a continuous and robust network connectivity
- Existing mobile resource discovery methods use a simple string-matching





# Main issues (1/2)

- In mobile environments, network consistence varies continuously: temporary disconnections occur frequently
- Traditional SDP performances are considerably decreased
- A flexible resource search system is needed to overcome difficulties due to the host mobility
- A **decentralized** approach is required
  1. A node should not be depending on some other node to advertise/register services
  2. Each resource should be autonomously exposed
  3. Applications on the other nodes of the MANET should be able to autonomously perform a discovery



# Main issues (2/2)

- **Syntactic matching** is largely inefficient in ad-hoc environments where there is not a common service interface
- Need to submit articulate requests to the system, to obtain adequate answers
- Need to cope also with non exact matches to:
  1. carry out resource discovery covering as much as possible the needs of requester
  2. grant also a partial satisfaction of the user request



# From UDDI to jUDDI (1/2)

- **UDDI** is a **platform-independent, XML-based registry** for businesses worldwide to list themselves on the Internet
- UDDI enables businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet. A UDDI business registration consists of three components:
  - **White Pages** — address, contact, and known identifiers
  - **Yellow Pages** — industrial categorizations based on standard taxonomies
  - **Green Pages** — technical information about services exposed by the business
- Designed to be **interrogated by SOAP messages** and to **provide access to Web Services Description Language documents** describing the protocol bindings and message formats required to interact with the web services listed in its directory
- UDDI relies on a vision of a world where Web Services would be linked up with providers through a dynamic brokerage system



jUDDI is an open source Java implementation of UDDI specification for Web Services

## jUDDI features

- Open Source
- Platform Independent
- Supports for JDK 1.3.1 and later
- UDDI version 2.0 compliant implementation
- **Use with any relational database that supports ANSI standard SQL (MySQL, DB2, Sybase, JDataStore, HSQLDB...)**
- Deployable on any Java application server that supports the Servlet 2.3 specification (Jakarta Tomcat, JOnAS, WebSphere, WebLogic, Borland Enterprise Server, JRun...)
- jUDDI registry supports a clustered deployment configuration
- Easy integration with existing authentication systems

## BASICS

- Is based on 128 bit UUIDs
- Each ID is associated to a specific service class
- Bluetooth SDP is code-based
- Only exact matches are supported

## GOALS

- Search and retrieve resources whose description cannot be classified within a rigid schema
- Compute a distance between request and each offered resource
- Maintain **original features** of the standard

## MEANS

- Exploit theoretical approach and technologies of **Semantic Web vision** (Ruta et al., IJWGS, 2006)
- RDF annotation of resources w.r.t. an **OWL-DL** ontology
- Use of **inference services** to extract new information

## OUUID (Ontology Universally Unique Identifier)

Unused classes of 128 bit UUIDs in the original Bluetooth used to mark each ontology



preliminary exclusion of supply descriptions that do not refer to the same ontology of the request

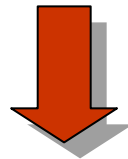
OUUID matching do not allow to identify a single service, but the context of resources we are looking for



can be seen as a class of similar services

- A generic client submits a request for a resource
- Provider-centric composition scenario
  - the provider collects various component resources coming from nearby hosts
  - the provider attempts to cover the request starting the composition algorithm
  - provider is exactly the service orchestrator and other nodes assume only a passive role
- Non exact matches: retrieved resources do not fill the request
  - an approximate solution is taken into account
  - an explanation of the approximation is provided
  - the requester could probably give up some part of the request

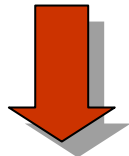
- Some services may set specific **execution prerequisites** to be satisfied
- if a device does not manage them, it cannot use the service/resource
- the composition requires to take into account **execution information**:
  1. inputs
  2. outputs
  3. **precondition** specifications
  4. **effect** specifications



we will consider only preconditions and effects

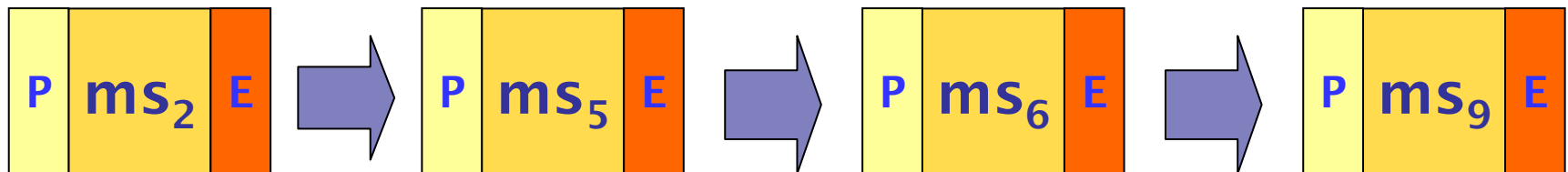
the approach can be trivially extended also considering inputs and outputs

- **Request:** a pair  $\langle D, P_0 \rangle$ , where  $D$  is the description of the requested resource and  $P_0$  are the provided preconditions
- **Mobile service:** a triple  $\langle MS_D, P, E \rangle$  where  $MS_D$  is the offered resource description,  $P$  the preconditions and  $E$  the effects

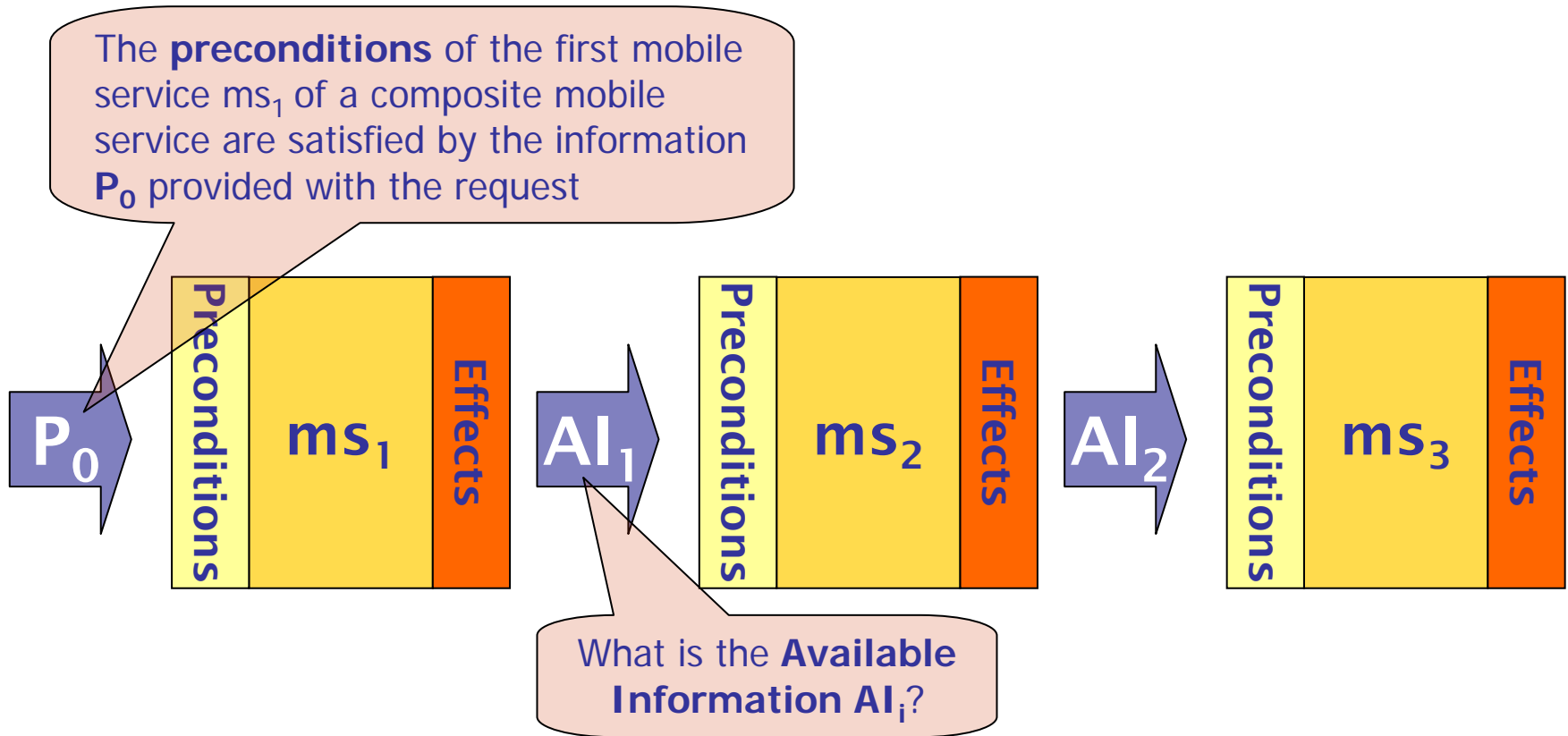


A **Composite Mobile Service (CMS)**:

- is not trivially a set of MSs:  $C = \{ms_2, ms_5, ms_6, ms_9\}$  preconditions
- is a **sequence** of MSs:  $C = (ms_2, ms_5, ms_6, ms_9)$



In order to execute a Mobile Service its preconditions must be satisfied, possibly using information provided by other MSs



$AI_i$  is the available information for the  $i$ -th mobile service  $ms_i$ . If  $E_j$  are the effects produced by  $ms_j$ :

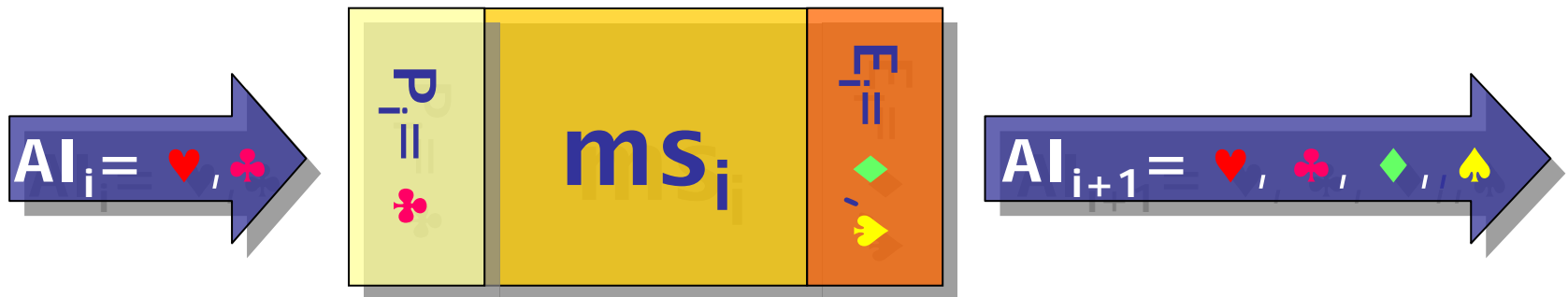
$$AI_i = P_0 \sqcap E_1 \sqcap E_2 \sqcap \dots \sqcap E_{i-1}$$

The Available Information  $AI_i$  must fulfill the preconditions  $P_i$ :

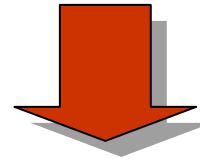
$$AI_i \sqsubseteq P_i$$

After the MS invocation the Available Information increase:

$$AI_{i+1} = AI_i + E_i$$



- In a pervasive environment, rarely all of mobile resources are simultaneously available:
  1. during the execution, a service could fail
  2. because the host is mobile, it could not be reached
- A MANET is continuously in evolution. While the composition is in progress:
  1. a better resource could be detected
  2. newer releases of an already discovered service could be now available

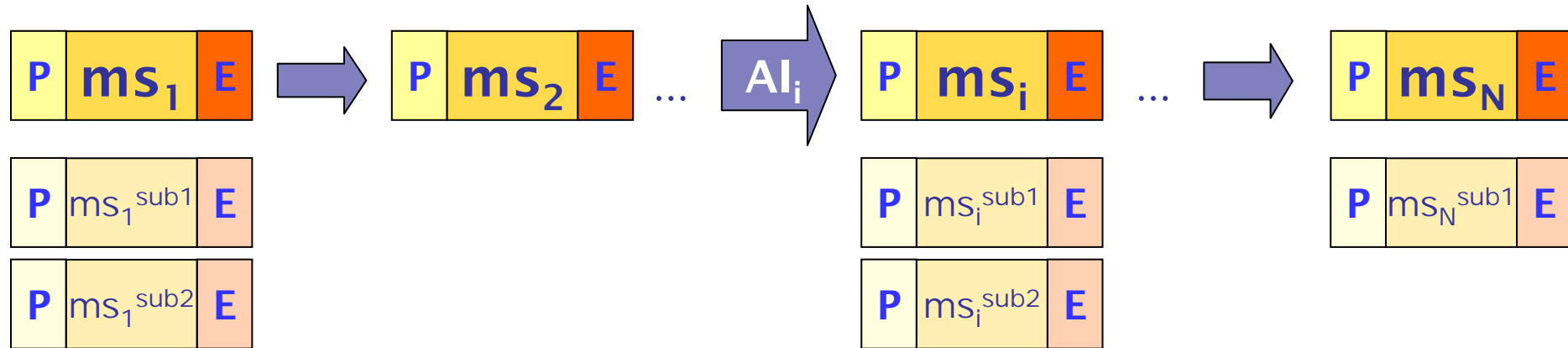


- the system should substitute mobile services no more suitable with new ones

DEF

**Similarity Group ( $SG$ ):** collection of component services which can be substituted with each other

- $\mathcal{SG}$  is created w.r.t. each resource in the  $CMS$ : each constituent service can have a set of substitutes



- In order to decide if a generic  $ms_i^{subX}$  belongs to the  $\mathcal{SG}$  of  $ms_i$ , given  $AI_i$  (Available Information up to the  $i$  service component):
  - calculate  $AI'_k$  ( $k = i+1 \dots N$ ):
    - $AI'_{i+1} = AI_i \cap E_i^{subX}$ ,  $AI'_{i+2} = AI'_{i+1} \cap E_{i+1}$ , ...
  - progressively check the satisfiability of the respective  $ms_k$  ( $k = i+1 \dots N$ )



# Framework interaction

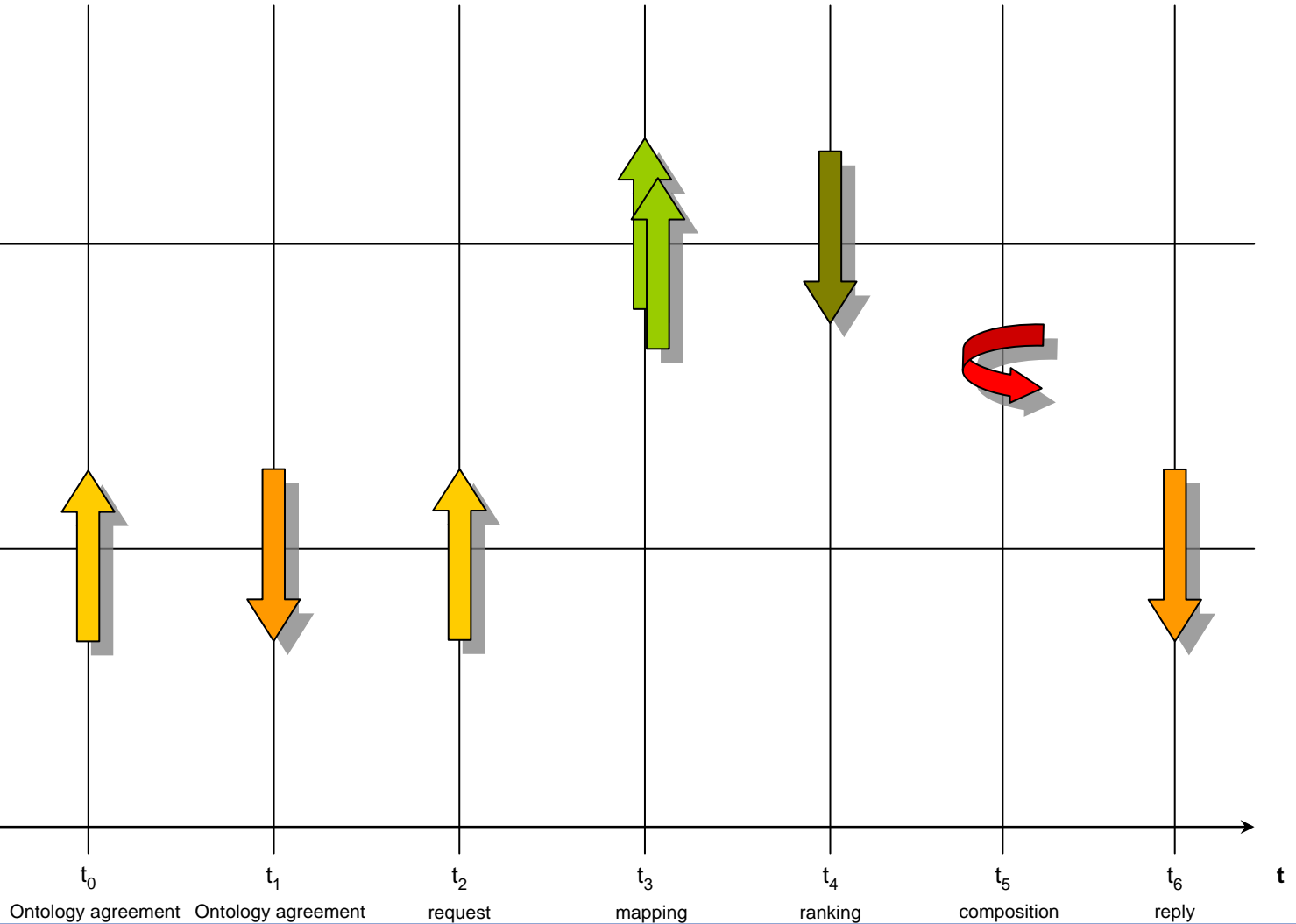
Service Selector



mobile hotspot



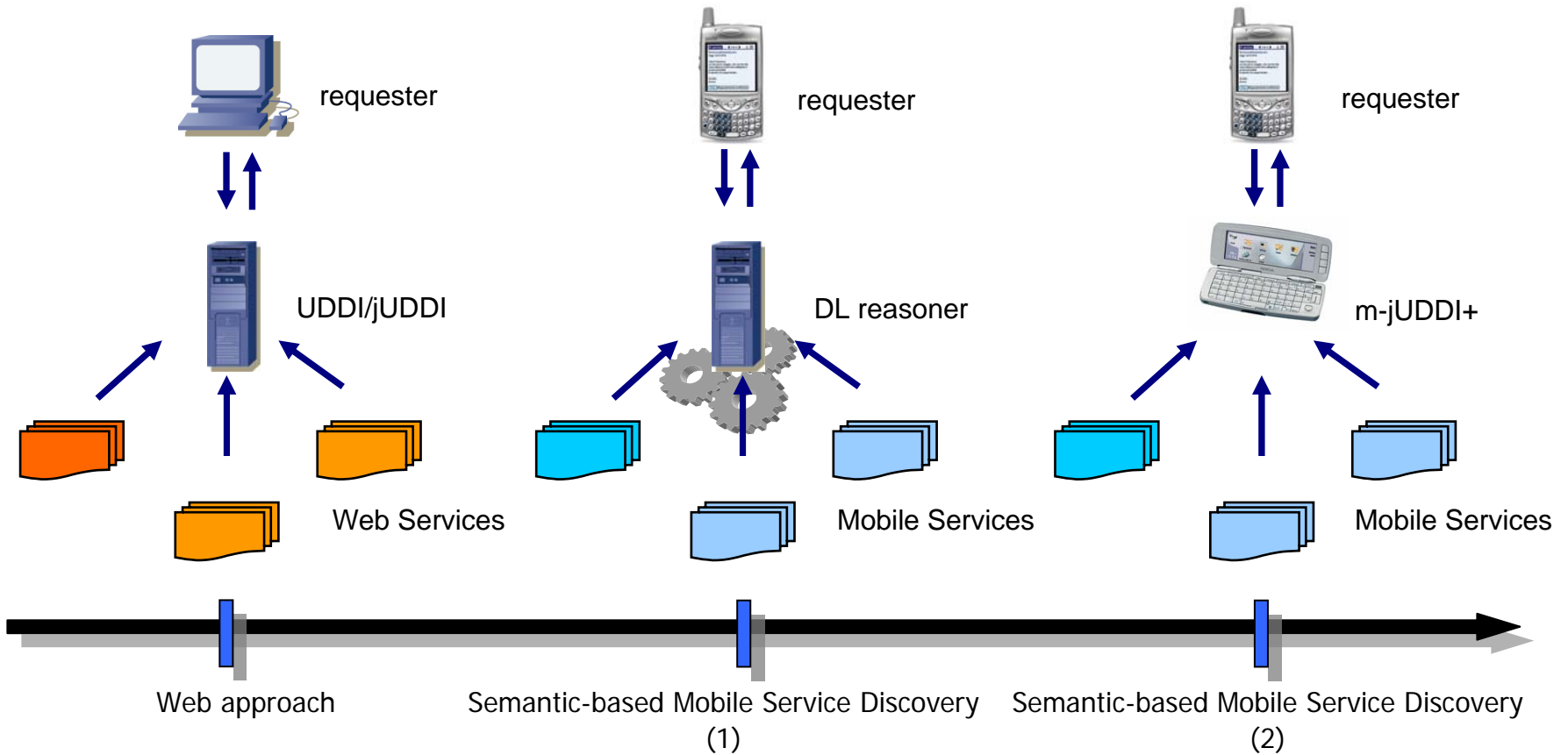
mobile host



- User and composer (*hotspot*) agree about the reference ontology  $\mathcal{T}$  (**ontology matching procedure**)
  - performed via the semantic-enhanced Bluetooth SDP
- Exploiting the selected ontology, the user formulates a request using OWL Classes and Properties in terms of:
  - required **preconditions**  $P_0$  and provided **effects**  $E$
- The *hotspot* selects service instances referred to  $\mathcal{T}$  and sends them to the Service Selector component ( $SS$ ), together with  $\mathcal{T}$ ,  $E$  and  $P_0$
- The overall information is mapped to the appropriate set of relational tables
- The *hotspot* with the  $SS$  provide:
  - the *CMS* also computing a matching degree
  - if an approximation occurs, an **explanation** about the uncovered part of the request



- compatibility verification between user provided **preconditions** and service preconditions
  - in case, the  $MS_D$  is selected as a candidate to satisfy part of the request
- compatibility verification between provided **effects** and required ones
- ranking of compatible services
  - in order to evaluate its **degree of correspondence** w.r.t. the request
- $SS$  attempts to identify the *ms* better covering the whole request
  - in case of failure, it provides the service which better deals with the uncovered part of the request
- each found service is progressively added to the *CMS* and the uncovered section is progressively reduced for the following step
- such an iteration goes on until:
  - the request is fully covered
  - compatible component services are totally exploited
- what is still unfulfilled will compose the **rest  $H$** :
  - an explanation about the approximate solution to be provided to the user



- **Decentralization** of discovery and composition of mobile services



▪ The approach aims to avoid employing in pervasive frameworks computationally-demanding reasoners

- A generic XML schema of an OWL-S ontology only presents subsumption relations which can be mapped into a relational database collection
- To perform discovery and composition of mobile services, their functionalities are substituted -to some extent- with structured queries over a DB set
- Recall that the relational model implies -due to its intrinsic structure- the possibility to establish well known relationships among generic entities
- It can be correctly exploited to elicit new information starting from the ones stated within a specific model instance

- **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 via possible roles
- **Parents\_i** tables ( $i=1..N$ ) are built expanding all the relationships of order higher than first among concepts
  - **Parents\_i** is derived joining **Parents\_{i-1}** and **Parents\_0**
- **Ancestors** table is given by joining all the **Parent\_i** ( $i=0..N$ )
  - It 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

- Discovery procedure receives in input the list of request concepts in a hash-set. For each concept 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 used for selecting from *Normalized* table services containing 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 compared with the ones in the *MSF* and discarded if already inserted
- In case, for each candidate component service a rank value will be computed. Best matching service is selected and inserted within the *MSF*
- If no instances result suitable for enriching the composition flow, the process is halted

$$rank = retrievedConcepts \cdot \left( 1 - \frac{Not\_retrievedConcepts}{TotalConcepts} \right)$$

Ranking formula



The proposed approach has been implemented and tested over a HP iPAQ 2210h PDA

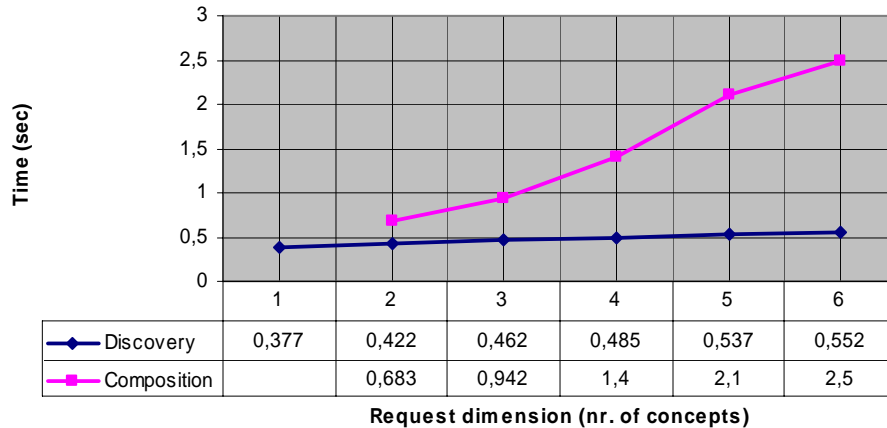
Two different ontologies have been used

- 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

Tests have been carried out evaluating:

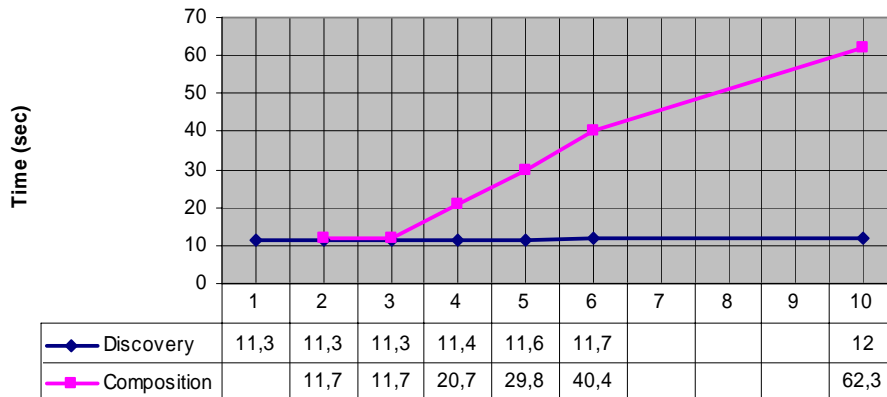
- **time progression** of discovery and composition procedures w.r.t. request dimension (in terms of component concept number)
- **time consumption** for the bootstrap phase of the application in both cases

KB 1



Request dimension (nr. of concepts)

KB 2



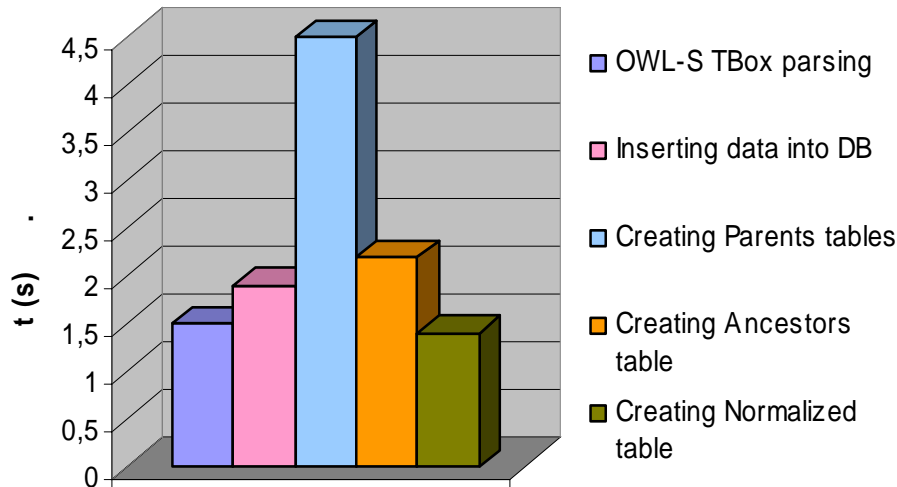
Request dimension (nr. of concepts)

The proposed system showed a better behavior - in terms of response time - w.r.t. a traditional reasoner over a wired server

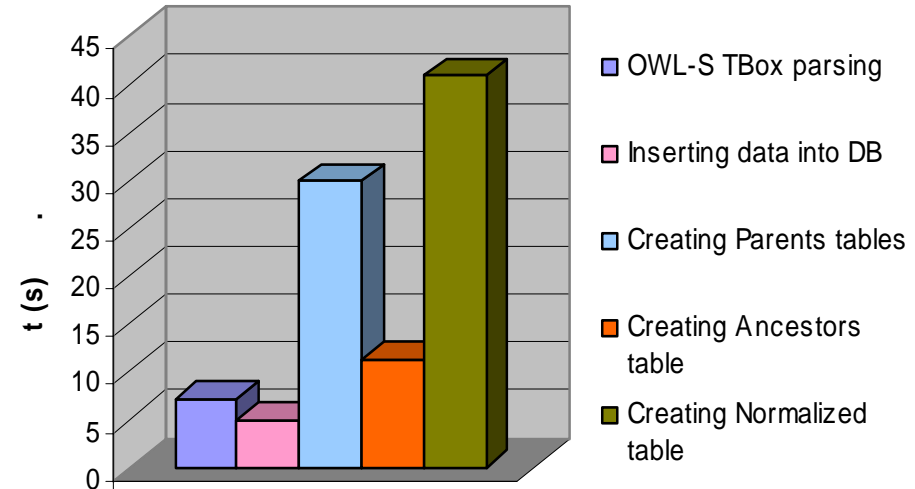
It has been noticed that response time noteworthy decreases when battery is only partially charged

All the carried out tests have been performed a fully charged battery for the PDA

KB 1

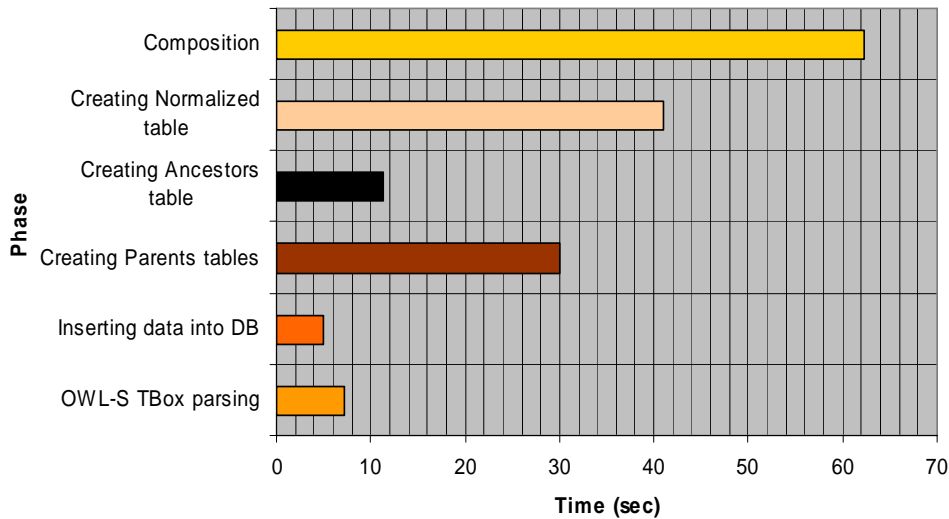


KB 2



A general comparison of bootstrap phase duration against composition one (computed in the worst case, *i.e.*, attempting a composition over a 10 concepts request exploiting Onto 2) points out that the initial processing is prevailing

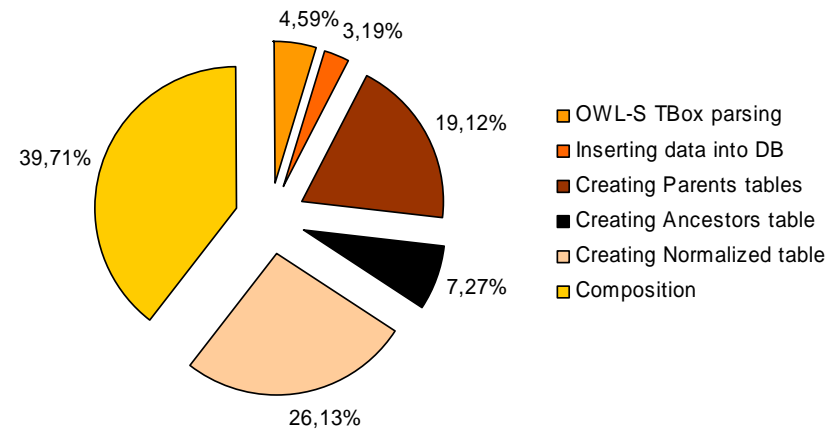
Time consumption comparison



Time consumption for mapping the KB into the DB is relevant (the creation of **Parents\_i** and **Normalized** tables takes up most bootstrap required time)

1. Mapping operations are performed only after ontology agreement
2. KB instantiation is not executed if not necessary

Time consumption comparison





# Conclusion and Future Work

1. We proposed a framework to enable the Semantic-based **service discovery**, **composition** and **substitution** in ubiquitous environments
2. The presented approach is based on adaptation of OWL-S compliant Knowledge Representation techniques and technologies to ubiquitous contexts
3. Main features of the proposed approach:
  - decentralized SD
  - non exact semantic matches management
  - support to substitution of failed or unreachable mobile services/resources
  - computation time saving
4. We are currently performing the porting of the approach under the open source SQLite environment
5. Involvement of emerging wireless technologies (RFID, 802.11, ZigBee) in a fully unified semantic-based discovery framework