Description and Matching of Services in Mobile Environments*

Johannes Grünbauer[1] and Michael Klein[2]

[1]Institut für Informatik, Technische Universität München,
85748 Garching, Germany, gruenbau@in.tum.de
[2]Institute for Program Structures and Data Organization, Universität Karlsruhe,
76128 Karlsruhe, Germany, kleinm@ipd.uni-karlsruhe.de

Service oriented computing is a new paradigm that is especially interesting in mobile environments. As a characteristics, functionality is hidden behind an interface and described as a black box with the help of a service description language. This enables participants of the network to enlarge the limited capabilities of their devices by using services provided by others. As service requestors and providers are not fixedly tied together but are dynamically matched and bound, this architecture is especially advantageous in mobile environments and their constantly changing situation.

Typically a service usage follows the so called *service triangle* (see Figure 1): The service provider (which can be mobile) wants to offer a certain functionality. He describes it as service using the service description language (Step 1) and publishes this description at the service repository (Step 2). This repository can be central or distributed. If a requestor (which can be mobile, too) want to use a certain functionality, he described his requirements as service request (Step 3) and sends it to the repository (Step 4). Here, a matchmaking of the request and the offers takes place (Step 5). Matching offers are returned to the requestor,

* The ideas for this paper have been developed at the Dagstuhl Seminar 04441 on *Mobile Information Management* in October 2004 together with Georgia Koloniari, George Samaras, and Can Türker.

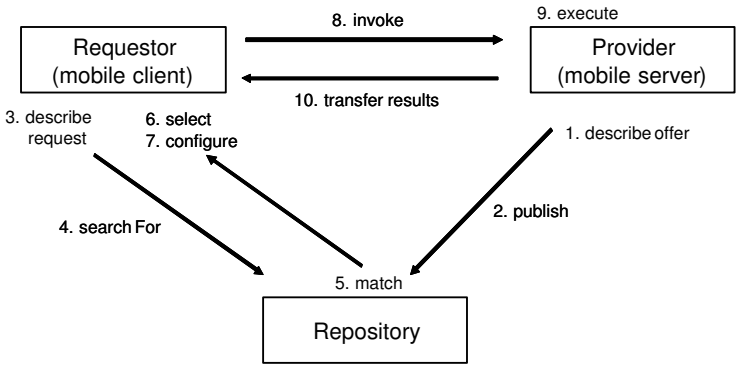

Fig. 1. The service triangle.

Dagstuhl Seminar Proceedings 04441
Mobile Information Management
http://drops.dagstuhl.de/opus/volltexte/2005/167

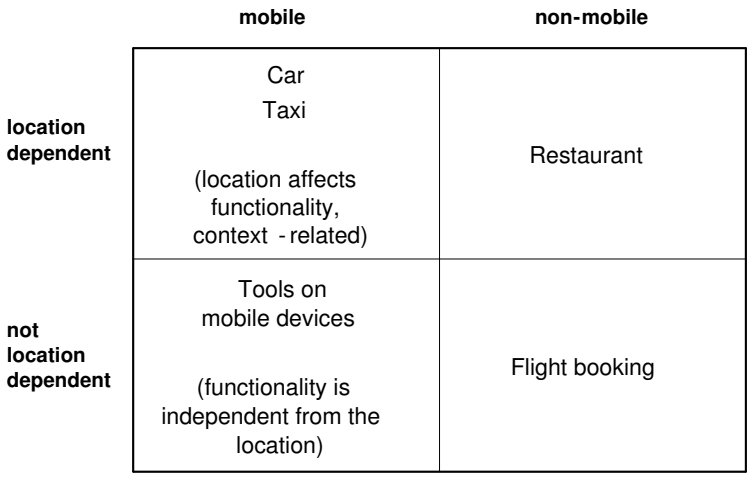

Fig. 2. Classification of services in mobile environments.

who selects one of them (Step 6), configures it according to his requirements (Step 7), and invokes the corresponding service provider directly (Step 8). The provider executes the service with the given configuration (Step 9) and – if there are any results – returns them to the requestor (Step 10).

In the following, we want to analyze three topics from this process with regard to their characteristics with regard to mobile environments: Service offer descriptions (Section 1), service request descriptions (Section 2), and the matchmaking (Section 3). We conclude the paper with a look on some challenges in this area.

1 Service Offer Descriptions in Mobile Environments

First, we give a classification of services in mobile environments. This can be done by splitting the space along two dimensions (see Figure 2):

- *Mobility.* Whether the service itself is mobile or not.
- *Location Dependency.* Whether the content of the service is dependent from the location where it is used.

The simplest services are non-mobile, non location dependent services like a flight booking service in the internet. It runs on a central server and provides a functionality that is not bound to a specific location. In contrast, a restaurant booking service could enable a client to reserve a place in a restaurant in a certain area. Therefore, this is a location dependent service. Examples for service where the providing device itself is mobile could be a taxi reservation service (location dependent) or tools like a dictionary on a mobile device (not location dependent).

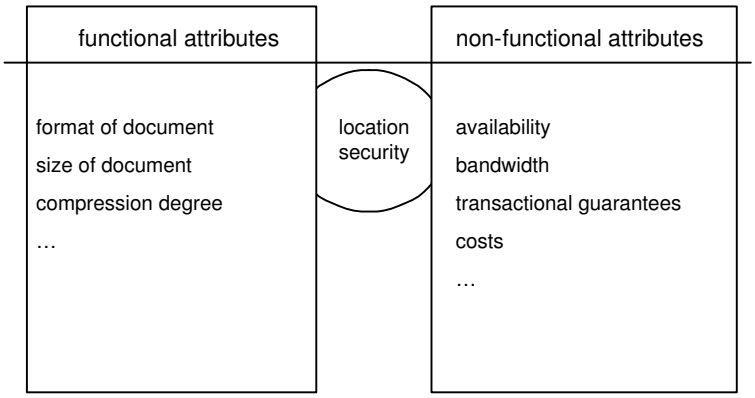

Fig. 3. Important functional and non-functional attributes for services in mobile environments.

When describing these services, it is necessary to give information about certain attributes that are of special importance in mobile environments. They can be divided in functional and non-functional attributes (see Figure 3). Important functional attributes could be the format, size, or compression degree of a file as mobile devices typically have limited capabilities in dealing with all file types. With regard to non-functional attributes, availability, bandwidth requirements, transactional guarantees, and costs could be relevant when checking whether a given service is appropriate in a mobile situation.

However, filling the attributes of a service description leads to problems in a mobile environment: Changes in the environment can also lead to changes in the description, so frequent updates of the description would be necessary. To avoid this, the description could be split up into two parts: a static part containing the regular service description and a dynamic part, which captures the current context of the service provider like his current location, the times of availability and so on. This division could be done logically only or lead to two different documents which could be stored in two different repositories. With our classification from above, mobile services would have a large dynamic part whereas non-mobile service would only have a small one.

2 Service Request Descriptions in Mobile Environments

Requesting services is quite similar to performing a database query. The user or a program needs to have a certain query language to get information from a service. In this section we take a look on the request descriptions in mobile environments.

In mobile environments there exists additional information which has to be given to the service–the context information (cf. Sect.1). The context adds additional constraints to the request. We call these constraints *implicit information*.

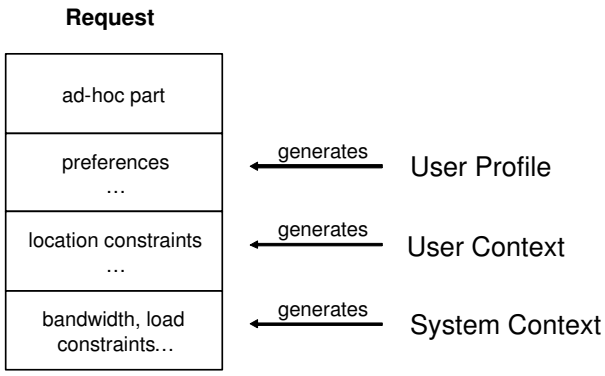

Fig. 4. Generating requests.

Example 1 (Taxi Service). A person queries a service to get a taxi ("I need a taxi within the next 20 minutes."). We have to distinguish between implicit and explicit information. The explicit information is, that a taxi is needed within the next 20 minutes. But this information is completely worthless if the service doesn't know *where* the taxi should be in 20 minutes. So, here the implicit information is the location of the user. □

Of course, there should exist the possibility to turn implicit information into explicit information since user should always keep his right of self-determination. So, if the user wants a taxi to be at a certain place (not at his current position), he should alway be able to generate a request with this information.

Example 2 (Reservation Service). Another example is a service, which can be used to reserve a table in a restaurant ("I want to reserve a table in a restaurant."). The implicit information for the service request could be:

- restaurant should be nearby
- restaurant sohuld be available
- the user prefers Chinese over Italian food. □

As shown in Fig.4, a mobile request is assembled by different parts: The *ad-hoc part*, the *preferences*, the *location constraints* and the *bandwidth and load constraints*.

The *ad-hoc*-Part is the request generated by the user. This is the explicit information the user gives to the service ("I need a taxi within the next 20 minutes" or "I want to reserve a table in a restaurant.").

The implicit information is generated by different aspects:

- The *preferences* are generated by the user profile. This profile can be set up manually or be generated by the user's behaviour over a period. Like in the example shown above, it could contain information like "user prefers chinese food" or "user likes musical shows".

– The user context generates the *location constraints*. This makes sense in case of using a PDA with a travel guide system which can e.g. offer information about sights nearby the user's current position.
– Furthermore, the system context generates *bandwidth constraints*. Let's go back to the travel guide example and let's assume, the user wants to watch an information movie about a sight he is currently standing in front of. If the bandwidth is low, the system should provide the user a movie with a low resolution or sound with a low quality.

3 Matching Service Descriptions

One of the most challenging part is that of the service matching. If the user starts a request for a service, there are different ways how he gets a result for the request. Figure 5 shows two extreme approaches.

1. On the left hand side the manual selection is shown. The user gives a simple request to the matcher. The matcher creates a list of services using a built-in heuristics and returns it to the service requestor. The user has now to pick one result manually and choose by himself which one he likes best. If he doesn't find a service he likes, he has to refine the request and start another search. This is the behaviour of common internet search engines, like *google* or *altavista*.

   As an example, the request could contain search words like

   request: {subject:taxi, time:18:20–18:40, location:plaza-hotel}
2. The other approach (right hand side of Fig.5) is the automatic selection of a service. The input to the matcher is a expressive and rich request description. That means, the request is combined by the user input data and the input given by the system context. The matcher now finds out which service is the best for the user and return exactly *one* result.

   Example:

   request: $\underbrace{\text{subject:taxi, time:20min}}_{\text{user input}} \oplus \underbrace{\text{location:plaza-hotel, time:18:20–18:40}}_{\text{context input}}$

In mobile environments, both matching types are thinkable. For ad-hoc queries, the first type is used because of its simple request descriptions; within mobile application, the second type should be used in order to ease the usage of the program.

However, both extreme approaches are not completely brilliant. The problem is, that in the first approach requires too much user interaction, and maybe the user has to refine his request a several times until he gets a suitable result. The problem with the second approach is that the matcher is too restrictive and takes away the user's right of self-determination.

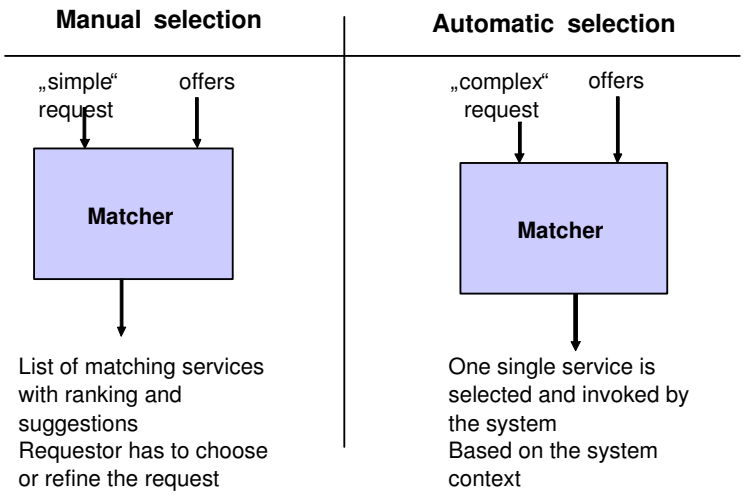

Fig. 5. Two extremes of matching services.

4 Further Challenges

In this paper, we analyzed the requirements and approaches for describing and matching services especially in mobile environments.

The following topics have not been addressed and can be seen as further challenges in the area:

- How detailed should be the service request? What constructs are needed? Are simple conjunctive queries sufficient?
- How to efficiently provide a list of ranked matches? What similarity measures are reasonable?
- How to refine requests using relevance feedback?
- How to monitor the system context during service execution?