Interaction Styles for Service Discovery in Mobile Business Applications

Markus Aleksy  Colin Atkinson  Philipp Bostan
Thomas Butter  Martin Schader

University of Mannheim, Germany
{aleksy|atkinson|bostan|butter|schader}@uni-mannheim.de

1 This work was supported by a grant from the Landesstiftung Baden-Wuerttemberg to Philipp Bostan and Thomas Butter.

Abstract
As the power of mobile devices continues to grow, and the range of resources accessible via wireless networks expands, there is an increasing need to offer services to users in a customized way, based on their immediate desires and context. At the same time, to construct such applications in a cost-effective and reusable way, there is also a growing pressure on mobile application developers to structure their systems in terms of a service-oriented architecture. However, these two goals are not always compatible. In this paper we present a new set of architectural components and principles which allow context-sensitive, mobile business applications to be assembled in highly flexible and reuse-oriented way based on the principles of SOA. We present the four main configuration patterns and interaction styles which this architecture supports and evaluate their pros and cons from the perspective of different infrastructure and usability issues such as bandwidth usage, latency needs, pricing and, privacy. Finally, we discuss which configuration to use in which circumstances.

1. Introduction

One of the major goals of mobile business applications is to deliver added value to the mobile user. This presents many challenges, but one of the most significant is the dynamic adaptation of a mobile application’s behavior to a user’s immediate situation. This implies that the services offered to a user need to be customized to his or her current context and desires. Context is any information used to deliver a service which is not explicitly input by the user. It can be any data about the immediate situation of the user and can be used to infer the desires of the users by assuming that specific tasks are bound to specific situations.

Finding suitable services to match a user’s context in an efficient and scalable way is a major challenge given the technical limitations of today’s mobile devices and wireless networks. These limitations include computation power, memory constraints, limited displays and low input capabilities. Although the computational power available in mobile devices has increased significantly over recent years, memory constraints still prohibit complex technologies like speech recognition or extensive data processing for example. Also the resolution of displays on mobile devices makes the representation of complex service results difficult.

Successfully addressing this challenge requires a flexible system architecture which is not only able to rapidly discover appropriate context-based services based on a user’s immediate situation and desires, but is also able to interact with the user in a variety of different ways depending on his/her situation and sophistication. The architecture must also be able to flexibility accommodate the wide range of existing service directories and brokerages.

To address this need the Mobile Business Research Group at the University of Mannheim has developed a generic software platform for location- and context-based mobile business applications, that allows different interaction styles and repository federation architectures to be configured with ease and speed. The architecture uses location information, calendar data, user profiles and other implicitly available contextual information to provide context-sensitive service retrieval and to enrich the information supplied to users. Since most services are only useful in certain situations, the challenge is to find the most suitable ones within the current context. This might for example be the time of day, location or even the available battery power [1].

In this paper we discuss the interaction between mobile client applications, service brokers and service providers, with a particular focus on the distribution of these entities in different architectural service discovery configurations. We use a novel approach which uses the same core components in different parts of the architecture depending on the specific configuration deployed.
2. Context-Driven Service Discovery Architecture

Mobile users have an interest in obtaining access to the services which, in their given environment, best meet their needs or desires. From the perspective of the user, this can be characterized as personalization. Usability and personalization are even more important for mobile applications than for traditional web-based services [10]. Discovering useful the most appropriate services for a given context, services require a lot of information describing the user’s current situation or task. Even if it were possible, given the limited input capabilities of today’s mobile devices, enter all this information by hand would be extremely cumbersome. To overcome these restrictions and avoid intensive user-application interaction it is possible to enrich the user input with information implicitly derived from context sensors. Using this additional information in the service discovery process considerably enhances the relevance of the search results in relation to the user’s current situation [2].

2.1. Entities of the Context-driven Service Discovery Architecture

Our service-oriented architecture for the context-driven service discovery contains the following components:

- **The Mobile Client** component is a client application, built using the SALSA client framework, running on a mobile device. The framework is responsible for mediating the communication with service brokers and service providers, and handles most of the context sensing. Data like the position, calendar information, device capabilities and profile data is gathered automatically. The mobile client offers a generic interface to the mobile user to query services and for displaying query results in the form of service descriptions. The client application adds implicit client-detected context information to the user’s service discovery request if available and explicitly allowed by the user.

- **The Service Discovery Service** (SDS) is a service broker that stores detailed descriptions of services that have been explicitly registered in its service repository. The SDS is able to retrieve services matching the users explicitly specified requirements augmented with context information from the mobile client and additional context provisioning services. In every architectural configuration there is always one first-level SDS that we call Universal Service Discovery Service (USDS). The USDS is configured as the bootstrapping SDS for the mobile client and is the first component queried for suitable services. Since the SDS technology is based on the composite pattern, the services returned by the USDS can be other SDSs that act as further service brokers, e.g. a gastronomy guide where restaurants are registered as services. SDS’s can thus be nested in arbitrary ways to best exploit existing repositories or service providers.

- **The Service Orchestration Engine (SOE)** is a control component that coordinates the interactions with multiple, lower-level SDSs in a nested architecture. SOE’s usually exists within SDSs but they can also be contained by client applications. A SOE coordinates service invocations and returns their merged results according to the Facade Pattern [4]. Additionally the SOE is able to invoke any service as a proxy for other components.

- **Service Providers** offer a service which may be used by any service requestor. A service may be an electronic service, e.g. a web service based gastronomy guide or a non-electronic service, e.g. a bar or café. Service providers may furthermore offer lower-level service brokers that offer a certain kind of service using SDS technology, for example the gastronomy guide. An SDS is also a form of service provider.

- **Context Provisioning Services** (CPS) deliver implicit contextual information or they derive new contextual information from existing information for the service discovery process. These services can be integrated into the mobile client application or the SDS.

2.2. Basic Configuration

The basic configuration of our service discovery architecture follows the principle of service-oriented architectures represented by a triangle between the service requestor (mobile client), service broker and service provider. This configuration is described by the Lookup pattern [7].

Service providers are registered at an SDS. The mobile client sends a service request augmented with available context information from client-side CPSs to the SDS which can add further context from server-side CPSs that are available. The SDS returns the most suitable service descriptions corresponding to the request and context from its repository to the mobile client. Finally, the mobile client directly invokes one of the returned service providers using the service descriptions from the SDS. If the chosen service is an electronic service it is invoked over the Internet and the user receives the delivered value.
3. Configurations and Client Interaction Styles

In general, there are several different possible ways of organizing service brokers within a service discovery architecture. In [11] they consider the organization of service brokers by service category (e.g. the NAICS taxonomy [12]). For architectures used in conjunction with context-based services, where location plays a major role as context, service brokers can be organized by location, where descriptions of services available for a certain region are stored in a broker responsible for that area. There are several published federation approaches which federate service or context discovery by region in this such as for example NEXUS.

The overall organization of SDSs in the various approaches retains the basic principles of [11] where lower-level SDSs are specialized service brokers that focus on services of a certain domain.

Using our SDS technology different configurations can be configured in which an SDS may act as a proxy to other services, return service descriptions directly or return pointers to other SDSs which are better able to deal with the given client query.

The four main configurations are referred to as: User-Managed Linear Configuration, Client-Managed Linear Configuration, Client-Mediated Linear Configuration, and Server-Managed (Hierarchical) Configuration.

The Composite Pattern [3] plays a key role in all of the configurations. Each SDS may contain other SDSs. Or stated differently, the services contained within an SDS may themselves be SDSs. This enables SDSs to be combined or federated in arbitrary ways. As mentioned in section 2.1 there must always be a first-level SDS, called the USDS, which appears in all of our configurations, but apart from this requirement, arbitrary configurations can be constructed.

3.1. User-Managed Linear Configuration

In the User-Managed Linear Configuration, shown in figure 1, the user submits his desire to the client software that acts as the service requestor and sends requests to the USDS. The USDS as service broker returns a list of service descriptions of lower-level specialized SDSs which are presented to the user by the client software as services. The user then chooses an SDS according to his personal preferences and trust in the provider. The request is sent again to the chosen SDS which returns a suitable description of a service provider that is presented to the user by the mobile client application. This interaction could also be performed over more than two levels since lower-level SDSs may return further specialized SDSs.

3.2. Client-Managed Linear Configuration

Figure 2 illustrates the idea of the Client-Managed Linear Configuration, that takes the responsibility of choosing between the USDS returned service descriptions from the mobile user. The service descriptions that are returned by the USDS are processed automatically by the SOE of the client application, which sends requests related to the user’s initial requirements to all listed SDSs and returns to the user a merged result set. In this case the client application’s SOE has to detect duplicates which could occur from similar SDSs and filter them. This can be difficult due to the restrictions and limitations of mobile devices, primarily due to the multi-source nature of the data [8]. Furthermore the client application has to connect to an unknown number of lower-level SDSs over the wireless channel.

3.3. Client-Mediated Linear Configuration

The Client-Mediated Linear Configuration, represented in figure 3 is similar to the User-Managed Linear Configuration described in the previous subsection. The difference is that the mobile client application is customized by software components downloaded from service providers and executed by the component
4. Limitations and Capabilities

Each presented configuration type shown has different strengths and weaknesses. In Table 1 a summary of these for each configuration type is shown. In this section we will discuss these tradeoffs in more depth.

4.1. Bandwidth Usage and Latency

Bandwidth is still quite expensive today for most wireless cellular networks and should therefore be taken into account when choosing a configuration. Furthermore the latency imposed by wireless networks is quite high compared to wired networks. Any configuration type needs to limit the bytes transferred over the wireless network and should try to offer a good user experience even with high latency. The Server-Managed Configuration obviously delivers the best results from the point of view of bandwidth demand. Since most of the communication in the service discovery process is handled between SDSs in using fixed networks, the use of the wireless channel is minimized. In this configuration, result duplicates are filtered on the servers and do not have to be transmitted to the mobile client. In many cases it may not be necessary to query a lower-level SDS. If so the User-Managed Configuration could save bandwidth, since it avoids unnecessary communication to SDSs involving the user in the selection process. The Client-Managed Configuration is the worst solution from the point of view of bandwidth usage, since all the results from lower-level SDSs must be transmitted over the wireless channel to the mobile client application’s SOE so that the filtering can be performed locally on the mobile client. The Client-Mediated Configuration that downloads components to the mobile client application makes similar demands on bandwidth as the User-Managed configuration. But it also uses additional bandwidth for component downloads. Storing these on the mobile device for further requests when services are reused is essential for effective usage.

Latency is another very important factor in network communication. Here the Server-Managed Configuration offers the best result since only one wireless round-trip interaction is necessary when a service is requested. The Client-Managed and Client-Mediated Configurations need several client-server invocations that lead to addition of latency for each single invocation. The Client-Managed Configuration will need the most extensive client-server communication, since all responsible SDSs are queried for suitable services. The client could try to alleviate the impact by sending the queries simultaneously.
4.2. Mobile Devices

PDAs and smartphones are used for many similar purposes today. Nevertheless, there are many between them, particular with regard to their input and output capabilities. When using a smartphone for service discovery, a quick answer with as few input steps as possible and an adequate representation of the service result is needed. The Server- or the Client-Managed Configuration can deal with these requirements. Also the User-Managed Configuration is useful under these circumstances.

The larger display and better input capabilities of PDAs offer much easier query input, while the extra memory resources make it possible to download components. This allows more complex user-application interaction. In this environment the Client-Mediated Configuration is preferable since it offers better customization to specific services.

4.3. Java environment

The Java2 Micro Edition (J2ME) [4] is the dominant platform for mobile devices. There are two different configurations available for mobile, the first and far more widespread version being the Java2 Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) [5].

The second configuration is the Connected Device Configuration (CDC) [6] which is aimed at high-end devices. While CDC is very similar to Java2 1.3, CLDC lacks many important features. Although it fulfills the requirements of the User-Managed, Client-Managed and Server-Managed Configuration it is not able to accommodate the Client-Mediated Configuration. In this case, the reflection mechanisms of CDC are required to support the downloading and execution of the service customized components.

4.4. Pricing

In a SOA, similar services could be offered by different service providers at different rates. Users want to have transparency of the accounting techniques used and need to know how much money they are spending for a service invocation or for service discovery. With Server-Managed or Client-Managed Configuration the user is not able to select the services that are used during the service discovery process. The only way to influence the pricing would be to set an upper bound for expenditure and let the SOE choose which services to use within the specified bounds. User-Managed or Client-Mediated Configuration allows the user to decide on his own if the price for an invocation or service discovery is reasonable for each service. He can base his decisions on his trust in a specific service provider or service brand.

4.5. Accuracy and Completeness

The configuration types have different characteristics regarding the accuracy and completeness of results. The User-Managed configuration only queries one chosen lower-level SDS and may miss some better results which might be provided by a different lower-level SDS. If the user can not decide which lower-level SDS delivers the optimal result this configuration is clearly inferior. The most complete set of results is provided by the Client-Managed or Server-Managed configurations since all lower-level SDSs are queried and all results are merged. This process of merging service results involves the elimination of duplicates which could lead to a decrease in accuracy since some results might be accidentally deleted. If the duplicate elimination algorithm is reliable, however, this configuration is superior.

4.6. Privacy

Privacy implications are an important factor in mobile business applications [9] and should be considered when choosing a configuration type. In the case of the Client-Managed and Server-Managed configuration context-enhanced query data is sent to many SDSs by the SOE without user intervention. Here it is automatically presumed that the user trusts any lower-level SDS. In the case of the User-Managed and Client-Mediated configurations, on the other hand, the user is able to choose which SDSs or service providers he trusts and is able to specify whether he wants to send private or sensitive information to them. This is especially important if lower-level SDSs may register themselves at a first-level SDS.

4.7. Client Software Complexity

Considering the size available for software on mobile devices and the cost of reboots in the case of software errors the complexity of the software needed is an important factor. With Server-Managed Configuration a lot of the query logic is situated at the SDS and the software needed on the mobile device is limited. This drastically improves maintenance and the update process. In any case, a universal user interface is needed that supports interaction with any service based on a generic service description model. Returned service results need to be prepared dynamically for presentation. The Client-Mediated Configuration shifts this issue to customized small software components downloadable from each service provider.
4.8. User Experience

The user experience is also an important factor. Given the restricted input capabilities of mobile devices the user-application interaction should be reduced to a minimum. Furthermore, the input needs to be accepted in a simple user-comprehensible format. This clearly favours the Client-Mediated configuration approach since it enables services to adapt the user interface to their specific domain and to specific situations described by the context. The Server-Managed Configuration allows very easy usage which might be preferred on mobile phones, but most users want to keep control over the actions taken by their device. User-Managed Configuration offers this control but might be too complicated for many users and most tasks.

4.9. Scalability

Scalability is an important issue because the number of users of an SDS may change dynamically. By federation of SDSs in the Client-Managed, Client-Mediated and User Managed Configuration the load on a single SDS can be reduced. SDSs can be federated along geographical or service types as mentioned in chapter 3. With the Server-Managed Configuration the SDSs may still be federated, but all requests have to be routed through at least the USDS. The best configuration for scalability depends on the depth of the federation. If the user gets a lot of lower-level SDSs returned, a new bottleneck is created on the mobile client and only Server-Managed Federation is feasible.

5. Conclusions

This paper has presented a highly-flexible, context-driven service discovery architecture and explained the configurations in which it can be used. We have evaluated the pros and cons of each configuration and explained their interaction characteristics from the client perspective. In section 4 we also discussed the characteristics of each configuration from the perspective of different requirements, summarizing the tradeoff options in table 1.

For the SALSA project, which focuses on scenarios in which the user wants to find a previously unknown service in an ad hoc way depending on the current situation, we have identified the optimal configuration as the Client-Mediated Configuration, since we rely also on CDC as the mobile client programming platform. This configuration has only minor drawbacks and it offers a very good user experience, better offline functionality and a better user-service interaction. In the event that CDC is not available the choice depends on other factors. In particular, it can not be assumed that users will trust all SDSs or service providers - a fact that makes Server-Managed Configuration an unsuitable choice. In general, the requirements and needs of the scenario in hand will determine which configuration is the most suitable. The key advantage of the architecture is that the basic components can be used to support any configuration in a flexible way to best meet the prevailing requirements and environment.

References


Table 1: Limitations and Capabilities

<table>
<thead>
<tr>
<th></th>
<th>User-Managed</th>
<th>Client-Mediated</th>
<th>Client-Managed</th>
<th>Server-Managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Latency</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mobile Phones</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>CLDC</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>CDC</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Pricing</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Accuracy</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Completeness</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Privacy</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Complexity of Software</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Complexity for User</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Scalability</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>