Mobile Service Oriented Architectures (MOSOA)

Jilles van Gurp, Anssi Karhinen, Jan Bosch
Software and Application Technologies Laboratory
Nokia Research Center
P.O. Box 407, FI-00045 NOKIA GROUP, Finland
[jilles.vangurp|anssi.karhinen|jan.bosch]@nokia.com

Mobile services hold a promise of utilizing the phone also for other purposes than purely communication. However, repeated attempts at realizing mobile services in the market place have been met with limited success. This article (1) defines the architectural drivers that drive success or failure of mobile services, (2) analyzes three different architectural styles of realizing such a mobile service using the example of a movie ticket selling service and (3) presents the results of this analysis. The main result of the analysis is that a serious conflict exists between usability and essentially all the other architectural drivers included in our analysis, i.e. portability, deployability and scalability. This is due to the fact that, because of the restricted state of the art technology, only native client applications offer satisfactory usability, but these do not satisfy the other drivers.

Introduction

Mobile services hold a promise of utilizing the phone also for other purposes than voice and SMS communication. Turning the promise into reality has proven to be more complex than what was anticipated in the dawn of digital mobile communication. Offering highly usable, value adding services to consumers and enterprise users has challenged the technology developers, business developers and the mobile service concept developers. The mobile services field became divided into content services and added value functional services early on. The main function of content services is to deliver media content such as ringing tones, greeting cards and background pictures into the mobile phone. Such services were originally built on top of mobile messaging technology such as SMS and have achieved continuously growing success in consumer segment. A more challenging area has turned out to be the services that aim to provide end users with added value functionality such as, for example, route planning, e-commerce and mobile payment. Some of these services were originally built on top of mobile messaging technology and then later on a browser based approach using WAP technology. However, consumer acceptance and business model continued to be a challenge.

An example of technology that aims to deliver ready-to use services for consumers out-of-the-box is SIM-ATK (Application ToolKit). SIM-ATK works in GSM networks. It allows the operators to include pre-installed service menus in the SIM card delivered with the subscription. The service menus are automatically integrated

with the native menus of the phone for seamless access by the end-user. These services use the short message protocol (SMS) to implement access to server functions. This sets some limitations for the implementation of the user experience as the interaction between the user and the server is not synchronous. Success of mobile service technology seems to have depended much on the level of integration into the native phone user interface and the ease with which end users can access the additional functionality offered by the services. An important landmark has been the iMode technology originally launched by NTT-DoCoMo in Japan around the turn of the millennium. iMode offers end-users seamless integration of basic services such as email, weather forecasts, sports results, various content services, online banking, stock trading, online gaming and ticketing services. An important factor for the success of iMode has been the "always on" aspect of the services, which is based on mobile packet data technology.

As the processing capability of the terminal devices has kept growing and faster data protocols have been deployed in the mobile networks it has become feasible to closely emulate the user experience of a PC with fast Internet access. Some of the recent devices include HTML browsers that are capable of displaying dynamic HTML pages with interactive scripts thus bringing the idea of "Internet in a pocket" closer to reality.

Browser based services are evolving rapidly as the capability of the terminals is growing. Traditional problems that plagued the early WAP/WML based solutions where the appearance of a service had to be tightly optimized for mobile device through custom design are slowly vanishing. To publish an existing WWW/HTML service for WAP access typically requires either a separate implementation of the user interface of the service or heavy automatic adaptation by a gateway which in many cases leads results in usability problems for mobile users of the service.

Another approach to offer highly usable mobile services and applications is to open the phone software platform for user installable native applications. This approach is available for example on phones using Symbian platform. A user installable application has access to all phone resources and can integrate with the native phone UI with no restrictions. This enables the creation of mobile services through a client-server pattern that are similar to the native functions available on a phone. A challenge in this approach is to ensure the compatibility of the phone software platform and installed applications. Additionally, the application installation process itself can be more complex than in a pre-installed services or browser based paradigms described earlier. The contribution of this paper is that it provides a comprehensive overview of the architectural alternatives that are available for the implementation of mobile services and analyses the different trade-offs of them.

The remainder of the paper is organized as follows. The next section presents the problem statement, followed by a section outlining the design drivers. Subsequently, three mobile service-oriented architectural styles are presented and analyzed. Related work and a summary section conclude the paper.

Problem Statement

Different brands, run-time platforms, screen sizes, networks and many other factors all contribute to the fact that there is a large variety of mobile devices on the market. To provide the best user experience, software developers of mobile applications must specialize their software for specific devices in order to take advantage of device specific features or device specific implementations of common features. A number of problems with device specific software exist:

- The number of devices that can be supported by specialized software is much smaller than the total number of devices. Specializing software for device specific features in any way causes the potential target market to shrink.
- Devices are sold on the market only for brief periods of time and are generally replaced by new devices months or at most a year after their introduction on the market.
- Developing software for a group of similar devices with a common set of specific features requires testing the software on all those devices.
- Device specific software may be hard to port to other devices that do not support the device specifics.

The goal of mobile services is to allow users to access these services through a mobile device. That means that a mobile service consists of a *client-side* component and a *server-side* component. The server-side component offers the feature, the client-side component makes it available to the user.

Necessarily, features, and usability of those features, in the client-side component depend strongly on the capabilities of the client device. Unfortunately, as outlined above making the client component usable by introducing device specific feature dependencies limits the potential market for the server.

As discussed in the introduction, the conflict between usability on one hand and market share on the other hand underlies the limited success in the market of mobile services so far. Solutions in the market so far have either suffered from poor usability (e.g. WAP [16]) or had the problem that the potential group of users able to adopt the solution was only a (small) subset of the entire group of mobile device owners

Mobile service oriented architectures need to address the following goals:

- **Number of devices.** The service must be provided on a wide variety of mobile devices. The more devices it supports, the larger the market is.
- **Native features.** The service must make full use of native features. Native features add value to the phone and therefore to the services provided on that phone. Native features include both software (e.g. text input methods) and hardware features (softkeys, camera's, display resolution). Native features relevant to the service should preferably be used.
- **Time to market.** The service must have a quick time to market. It is important to reach the market before the competition.
- **Window of opportunity.** The service must not miss its window of opportunity. It is important to be able to target the devices the service is developed for as soon as these devices become available on the market. The devices are on the market for a relatively short period of time and the potential revenue of a service is constrained by this period of time.

- **Forward compatibility.** The service must be forward compatible with the successors of the devices it is targeting. Users of the service will want to continue using the service when they purchase a new phone (assuming the service is useful to them).

Any successful architecture solution for mobile services will need to address these goals explicitly to the extent that is technically feasible. We have the following reasons to believe that it is now possible to define such an architecture:

- **Device performance**. Moore's law [13] has gradually improved device speed, bandwidth and capabilities. These abilities may be exploited to provide an acceptable end user experience while meeting most of the goals outlined above.
- **Consolidation**. There is a growing set of common features supported across a wide variety of devices that may be of use when implementing mobile services. This common feature set is good enough for a wide range of applications so an increasing number of mobile services can be implemented for a (relatively) broad set of devices.
- **Market size.** Adoption of mobile devices in the market has grown exponentially. A large part of the world population now has access to mobile technology.

The problem in mobile service oriented architectures so far has been that existing architectures fail to reach all aforementioned goals due to the fact that they are conflicting given the current state of the art. In this article we evaluate three architectural alternatives against the goals outlined in this section. In the next section we translate these goals into architectural drivers.

Architectural Drivers

In this section we look at the architectural drivers that influence the design, development, deployment and ultimately the success of mobile services. In the next section we will compare three architectures and analyse how they are affected by the architectural drivers.

Usability

In order to be adopted by users, mobile services need to be usable. By usable we mean that:

- It must be easy for the user to find and access the service.
- It must be easy for the user to make use of the service (learnability, ease of use)
- It must be convenient for the user to make use of the service (performance, usefulness).

In practical terms this means that the service needs to be integrated with the mobile user interface because this provides users with the fastest route to the service and the best performance possible on the device. Access points to the service may be embedded in menus, associated with soft keys, etc.

Experience with mobile user interfaces has shown that it is important to minimize the number of navigation steps to particular features [12]. For example, access to the contact list or message overview is rarely more than two button presses away on most mobile phones. Less important features may be located deeper in the menu structure.

The second requirement benefits from integration with the mobile user interface for several reasons:

- User input is typically best facilitated using the 'native' capabilities of the device. For example, many phones include smart text input features that aim to minimize the amount of button presses.
- The user presumably already understands the native user interface so using the service through that interface is easier than through a different interface.

Portability

The potential market for mobile services is huge. World wide there are now billions of mobile devices in use. A large and growing percentage of these devices is internet enabled (i.e. equipped with a TCP/IP network stack, connected to a TCP network and equipped with software components such as browsers that make use of this capability). To capture enough market share, mobile services need to be available to as large a subset of these users as possible.

This is technically hard because differences between devices tend to be difficult to bridge. For example, screen size, number of supported colours as well as the number of keys tend to vary across devices. Furthermore, there are differences in device capabilities such as supported networks (GPRS, EDGE, UMTS, CDMA), add-ons (e.g. GPS), camera and connectivity (infra red, Bluetooth, USB, WLAN). Dependencies on such features need to be carefully considered because of portability.

Deployability

A potential obstacle for users of the service is the issue of client side deployment of software components. Installing software of any kind is something few users know how to do. Therefore, (additional) client side components of a mobile service, if needed, should be provisioned over the network. Any kind of installation impacts the size of the potential user population negatively. Similarly, software updates should be totally transparent to the user.

Scalability

If successful, the mobile service may be used often by a large number of users. The maximum for this is the amount of users that own a device that can run the client side component of the mobile service. The architecture needs to be able to scale to this amount of users. There are at least these factors to consider:

- **Business scalability**. The business model should not include human intervention (e.g. a person answering the phone) or any other activities that are resource intensive unless this cost can be accounted for in the total price of the mobile service for the user.
- **System scalability.** The system architecture needs to be able to scale to the number of transactions successful adoption of the mobile service in the market will cause. If millions of users use a service several times per day, that means that the server side component of the service will be processing tens of millions of transactions.
- **Client scalability.** The client side architecture needs to scale well from low-end devices to high-end devices. Any scalability issues with the client side architecture will affect the potential amount of users for the service negatively.

Three Mobile Services Oriented Architectures

To highlight the properties of different architectural approaches we use an example mobile service realized using three alternative architectural styles. The example service is a movie ticket purchase that enables the user to purchase tickets for him- or her-self and a group of friends. The service also automatically generates notifications of the time of the show and the title of the movie for each. The usage scenario with totally seamless experience would be as follows: 1) User activates "ticket purchase" service through the phone. 2) User selects "movie tickets". 3) User selects the title. 4) User selects the time of the show. 5) User selects the friends he wants to include in the purchase from the phonebook of his/her phone. 6) User performs the purchasing transaction and is automatically authenticated and payment information is directed to the user's already existing billing connection (e.g. credit card, phone bill, etc.). 7) Friends receive a notification and an entry in their phone calendars.

MOSOA 1: Client-server with native client
Using a native client application provides the greatest flexibility for implementing the user interface and integration with the phone features. Figure 1 presents a high level design of the sample service using the client server pattern and a native client based on, for example, Symbian.

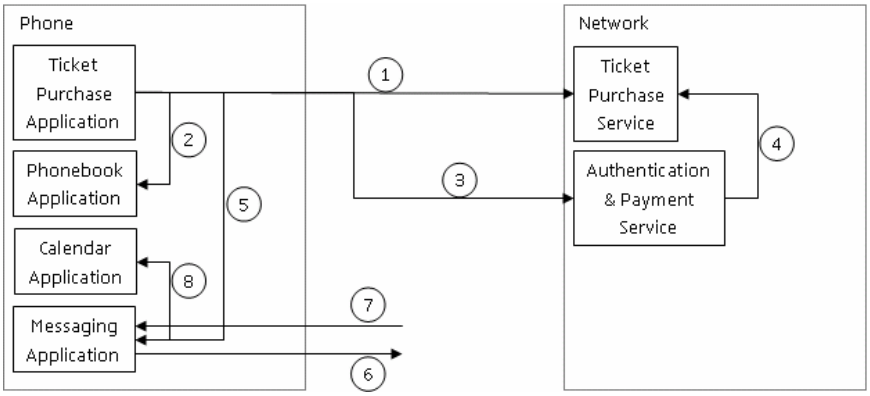

Fig. 1. A mobile service with a native client application.

The user is presented with a native ticket purchasing application that must have been installed into the phone in advance. The ticket purchase application in the phone knows the server part and the payment service. Steps one to four of our use case are performed by the ticket purchase application in the phone and the corresponding server part denoted by interaction "1" in the diagram. Ticket purchase service manages the catalogue of available movie titles. Next step of the use case involves interaction between different applications inside the phone. User can select the group of friends using the native phonebook application. This is denoted by "2" in the diagram. Authentication and payment information is managed by a service in the

network as shown by interaction "3". Information of the completed payment transaction is communicated to the ticket purchase service to complete the transaction, transaction "4". The client ticket purchase application then uses the messaging application in the phone to convey the notification and a calendar entry to the friends who were included in the purchase ("5" and "6"). Friends will receive a notification message with a calendar entry that the phone can automatically insert into the calendar ("7" and "8" respectively).

- **Usability.** Usability of a native client can be high since it provides the user with a native look and feel thus lowering the learning curve of the service. Typically, the native applications can be also tuned to effectively utilize the computing resources of the phone thus giving shorter start-up times and faster response times in the user interface. Integration with other native applications on the phone can also boost usability and the value of the service as we can see in our example. For example, it is simple for the end-user to keep one register of contacts and friends in the phonebook and use that asset for all communication and group transactions. Other good characteristics of native client applications include the possibility to support off-line functionality and special attached devices and special communication hardware like Bluetooth, IRDA and USB. Also the native client application can integrate directly to the power management system of the phone.
- **Portability**. Portability is a serious problem for native client applications. Only a small percentage of mobile phones support the installation of native applications. For the ones that support it, there exist different software platforms such as Symbian, PocketPC and Brew that are not compatible with each other. Also the native client application can depend on other native components that have to be present in the phone. This can create compatibility problems even inside one platform as the user must ensure that his or her phone includes all required components for the application to work.
- **Deployability**. Native client applications need to be installed using the native application management functions of the phone. Most platforms support different mechanisms to do this over the air (OTA), through USB connection or using a memory card. Application installation typically requires some technical ability from the end user and can thus be a major hurdle in the deployment of a new mobile service.
- **Scalability.** The native client server model scales well in performance as the resources of each client device are used partly in performing the service transactions. Requirements for the server side components are easy to isolate as the server transactions remain simple. Business scalability can be challenging as the deployability of native clients is rather complex. Communication costs from the native clients to the network services and other users, like the notification function in our example, can prove to be a difficult issue also.

MOSOA 2: Client-server with mobile Java client

The mobile Java client technology tries to avoid the portability and maintainability issues by providing a standard execution environment and deployment model for the client applications in a phone. The following diagram presents a high level design of our sample service using the client server pattern and a Java client application.

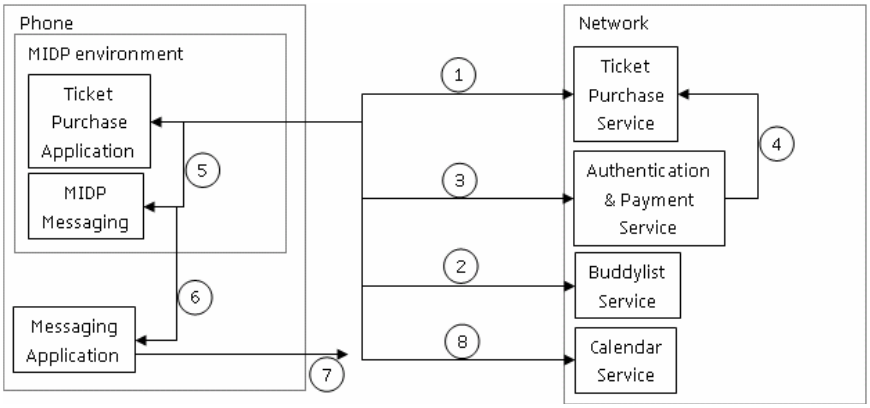

Fig. 2. A mobile service with a Java client application.

The MIDP environment for mobile devices is not actually one standard configuration. MIDP standard has two variants: the earlier MIDP 1.0 that is supported by most Java-capable phones and the recent MIDP 2.0 that is supported by latest models. Inside these main variants are different capabilities like messaging, full screen UI and encrypted communication which are only supported by certain phone models. These issues diminish some of the benefits that MIDP and Java were supposed to bring to the developers of mobile services.

In our example we assume that MIDP messaging functions are available to the Java client application. Steps one to four in the example service take place between the Java client and the ticket purchase server ("1"). Next step in the use case involves selecting the friends from a buddy list service that is hosted in the network in our case ("2"). The actual purchase transaction with authentication and payment happens in similar fashion as with the native client. Ticket purchase client communicates with the authentication and payment service which verifies the payment to the ticket purchase service ("3" and "4").

The Java client uses MIDP messaging to send SMS notifications to the friends involved in the transaction ("5", "6" and "7"). For setting a calendar reservation we assume a network hosted calendar service which is used by all users involved in our example case ("8").

- **Usability.** A Java client application can try to support native look and feel through the use of standard MIDP widgets but in most cases developers want to have more control over the UI and define their own UI elements. This can make the individual application quite usable but will make for steeper learning curve as the user has to adapt to application specific UI paradigms. Hosting a complete virtual execution environment in a phone will inevitably cause some overhead in application start delays and response times of the user interface. There are many recommended practices to optimize Java for mobile platforms, which can yield good results. Mobile Java makes it possible to integrate with some phone resources and native applications but not all. This can degrade usability as the user has to maintain contact information of his or her friends and calendar in many places like in our example.

- **Portability**. Portability for mobile Java clients is better than for native applications but is still a complex issue to tackle. Typically the developer has to balance usability and portability tradeoffs to come up with a solution that is supported with large enough device base and offers still well enough usability. The availability of several different standards for mobile Java including MIDP 1.0, MIDP 2.0 and CLDC 1.1 is a problem for portability. Many individual API standards for phone specific resources like messaging, call functions, phone lights and vibration are another problem area. Mobile Java offers ways to detect if a given feature is supported by the device thus making it possible to write client applications that can dynamically adapt to phones with different capabilities.
- **Deployability**. Mobile Java offers rather simple model for over the air provisioning of client applications. The installation process can be made easy enough for average end user to perform. Online updates can be initiated by the client applications themselves if needed.
- **Scalability**. Much of the functionality in Java client model is typically in the server side. The overall scalability of a service design thus heavily depends on the scalability of the server side. Business scalability can be good as it is possible to distribute and sell the client applications through WWW, mobile browsers and SMS rapidly covering wide user base for example through TV commercials with URLs or SMS instructions to install the service. Communication cost from the Java clients to the network services can be an issue depending on the underlying data communication business model.

MOSOA 3: Client-server with mobile thin client

The browser based approach has been very successful in the Internet and PC world. Initially the usability of services offered with browser UIs was poor and different browsers had serious compatibility problems but the emergence of "de facto" browser functionality and technologies like browser side scripting have improved the user acceptance of these services greatly.

Developing browser based services for mobile clients today is in many ways facing similar problems as developing browser based services for the PC in the nineties: Technology just hasn't converged yet. However, we can observe rapid advances in mobile browsers and it appears that we might actually leap a few generations compared to corresponding PC technology. Despite this, it is not possible to simply apply the user interface patterns from the pc world to the mobile world. The screen size requires different approaches to efficiently present information and allow the user to work with the on screen information effectively [12].

The following diagram illustrates a browser based design of our example service. Majority of the functionality of the service now resides in the network. Some native phone applications, like the messaging in our example, can still be utilized for services but they are integrated to the service through the network instead of locally.

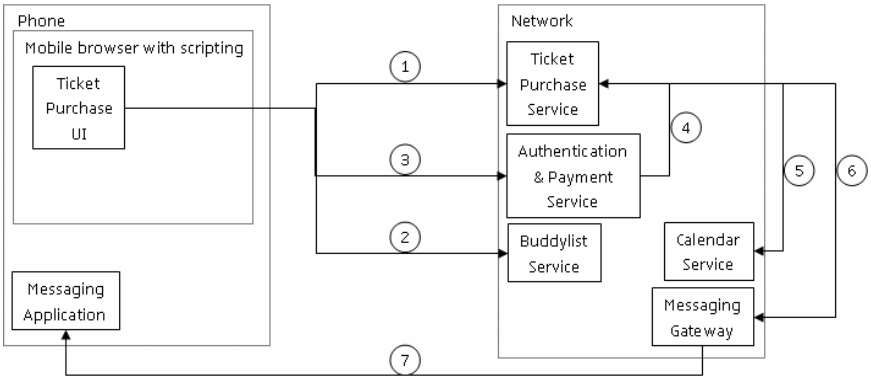

Fig. 3. A mobile service with a browser based client.

A browser based implementation of the example service must rely on the network side to provide the required service components. Integration to local phone resources from the mobile browser is currently very limited. The overhead to start using a browser based service is very low; it just requires the user to point the browser to the correct link to connect to the service. This can be facilitated through e.g. operator portal pages, SMS messages, TV commercials, etc.

For the first four steps of our use case the user interacts with the ticketing service through the browser ("1"). Selecting the friends to be included can be done from a buddy list service ("2") and possibly with some elementary data management within the browser, such as copy-paste through a clipboard (if that is supported in the device).

The actual purchasing transaction would follow the same general pattern as the earlier architectures ("3" and "4"). However, the next step has to be radically different from the client-server architectures. The ticket purchase service must be able to generate notifications and calendar entries to the list of friends ("5", "6" and "7"). All of the service components and data resources must reside in the network in contrast to the client-server models of previous examples. The user can still receive notifications using the native messaging application of the phone but it is not integrated to the user interface of ticket purchasing application thus degrading the user experience.

- **Usability**. Usability of an individual browser based service can be reasonably good for suitable applications. Typically, applications where the interaction between the user and the application is based on a forms paradigm fit well to the browser client architecture. Highly graphical user interfaces using direct manipulation paradigms are difficult to implement within a browser but emerging technologies like the AJAX will offer some level of support for it in the future. The main problem with browser based mobile service architecture is that it cannot use or integrate with the native applications and resources on the phone device. In practice the service components and resources used by an application must reside in the network. They can sometimes be integrated in the browser but the majority of the integration task must take place in the network. Another drawback of reliance on server-side components is that off-line operation is impossible: in order for the user to use the service the server resources must be accessible by the browser.

- **Portability**. Good portability is one of the strong points of a browser based architecture. Even so, the architect of a browser based mobile service is forced to select between many alternative technology stacks that are largely incompatible with each other. WAP/WML is the pioneer technology of mobile browser based services. In practice, it is currently being phased out due to the emerging of HTTP/HTML capable device generations. The actual capabilities of mobile browsers still vary considerably, however, and there is not yet a consensus on the basic capabilities similar to that in the PC world. New technologies like browser side scripting; AJAX (Asynchronous JavaScript And XML); XForms and HTML 5.0 are promising more interactive browser applications in the future but are rarely supported (even on PC based browsers) so far.
- **Deployability**. In principle the deployability of a browser based mobile service is as simple as it gets. It includes the normal deployment and configuration of the server side components and communicating the link (URL) of the service to the user population. Any user with a phone equipped with a browser that is compatible with the service can start using the service immediately. Any registration and authentication functions can be taken care of inside the browser session. Deployment can get more complex if a particular browser technology, like XForms, that is not pre-installed in the user's phone, is used. In this case the deployment phase is similar to client-server architectures.
- **Scalability**. The scalability in capacity of browser based architectures can be controlled by the traditional design solutions of the server side. This involves two main areas: the scalability of access to the service and the scalability of the communication between the service components in the network. The phone generally does not have much impact on the scalability of a browser based architecture except in special cases like when the amount of data sent to the browser exceeds its capacity (it does however restrict the amount of information that can be presented in a usable way [12]). The business scalability of a browser based architecture is very good. One can start with a low capacity server solution and increase the capacity gradually as the service becomes more popular.

Related Work

The notion of software architecture was popularized during the nineties. Perry & Wolf [15] in their article first identified the relevant concepts. Standardization efforts by the IEEE resulted in a recommended practice which has been widely adopted in the software industry [6]. Since the mid nineties, the notion of web service architectures has been popularized. The architectural style that underlies most web services, i.e. representational state transfer (REST), is a crucial ingredient of the hypertext transfer protocol which is also the transport of choice (though not the only one) for service oriented protocols such as SOAP or XMLRPC. In his doctoral thesis on REST [3], Fielding outlines the architectural principles that underlie the HTTP specification (of which he is a co-author) [7] and how these principles are crucial to providing scalability.

The World Wide Web consortium maintains a somewhat more narrow definition of what comprises a web definition. Their WS-Arch document [1], defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network". The definition is further constrained by specifying that the interfaces should be specified in a machine readable format (i.e. WSDL) and that other systems interact with the web service using SOAP messages.

For mobile services, this definition may very well be too narrow since handling SOAP on mobile devices is impractical due to the limited memory and processing capacity. However, REST principles as outlined by Fielding still apply to mobile service architectures. Scalability in the mobile world is of even more importance than scalability in the internet world because, especially in third world countries, there are vastly more people owning a mobile phone than there are people with access to PCs. The notion of local state is arguably of even more importance due to the inherently more unreliable network conditions (roaming between networks, areas not covered by any network).

A key difference between web service architectures and mobile service architectures is the client side. For web services, client side capabilities have a much higher degree of commonalities than is the case in mobile devices. The issues we outline in our problem statement mostly relate to the client side.

Some earlier work on mobile service architectures includes Hodes et al. [5], who propose an architecture for discovering and working, with local services. However, their work predates most of the internet services that have since emerged and consequently does not take these into account. Another article from this era by Jones et al. [10] analyzes WAP services from a usability angle. These articles are illustrative of the thinking on mobile web services in the late nineties. As described in our introduction, the approaches from this time (especially WAP) mostly failed in the market.

More recent work tends to focus on agents or semantic networks (e.g. [2]). However, such techniques are as experimental in the mobile internet as they are in the regular internet. While promising, we don't believe these approaches make explicit the requirements and challenges of mobile service architectures as we do in this article since they do not address all of the goals we outline in our problem statement section.

Some work has been done on evaluating conventional web service technology in mobile devices. For example, in [11], the authors evaluate the performance of SOAP over various protocols. This work suggests that the overhead of XML parsing is a major obstacle as is network speed. Additionally, some research has been done into using asynchronous messaging in mobile service architectures (e.g. [14]). Interestingly, recent developments in conventional web services also push towards an asynchronous style of working.

Additionally there has been a lot of effort evaluating usability of mobile user interfaces. For example in [16] the usability failure of WAP technology is analyzed. Our Nokia colleagues have written a comprehensive overview of usability issues in mobile user interfaces [12].

Summary & Conclusions

Table 1 summarizes the discussion from the previous section. The only quality attribute the native client scores best on is usability. This is probably the main reason why services with a native client (e.g. the ones discussed in the introduction) have more or less failed in the market. These services provided good usability on the devices where the client actually worked but at the cost of the other quality attributes in this table. Consequently lack of market share and the associated high cost of fixing that problem prevented the widespread adoption of such services.

	Native client	Java client	Browser
Usability	+	+/-	-
Portability	-	+/-	+
Deployability	-	+/-	+
Business Scalability	-	+/-	+
TCO	-	+/-	+

Table 1. Summary of architectural alternatives

As the table suggests, non-native clients are the solution to the problem. Unfortunately, this comes at the expense of usability. However, we also observe that this is increasingly less a problem as for example more Java MIDP APIs are standardized and deployed. Both the Java virtual machine and the browser are implemented as native application. In theory this means that they can provide the same level of usability as a native client. Features such as power management, user interface, input methods can all be put to use in the implementation of a browser.

Unfortunately, the above requires standardization of the way such features are exposed to the service application developers. Currently, there exist many standardized MIDP APIs for a wide range of mobile phone specific features. A continuing problem is that most of these APIs are optional so depending on such APIs is almost as bad as depending on native features directly.

Based on our experience with mobile device technology, we are convinced that a consensus on mobile device features is emerging and that consequently it will become easier to abstract from such features using e.g. MIDP APIs. The same has happened on the PC desktop where applications tend to be much more portable across different desktop platforms than is common on mobile platforms.

We have identified a clear trend that what we call a browser in this article will in the future evolve towards a standards based application container. In principle, given that proper standards emerge, this means that applications inside such a container can rely on features similar to those exposed in high end MIDP containers today. There are multiple ways of realizing such integration. An obvious way to this is through plug-ins. Additionally, features may be exposed through custom URL schemes like for example the tel://<phonenumber> [8] or the sms:// <phonenumber> [9] type schemes.

A second trend we have identified is that in future service application containers like outlined above, the notion of local device access will be of less importance than it is today. Resources will become themselves services. For example, there is no technical reason to have features such as contacts and SMS messages client side other

than optimizing access to these resources and ensuring that these resources are available when the phone is offline.

Ultimately, there will be this convergence of applications and services where the particular client used to access the service depends on the user context. For example the same list of contacts may be manipulated from an office environment on a laptop and from a contact list service application on a phone.

Technically, the service components would be hosted on a heterogeneous grid-like environment where resource allocation and provisioning of client components is automatically managed without user intervention.

References

[1] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard. "Web Services Architecture", Web Services Architecture Working Group, http://www.w3.org/TR/ws-arch/, February 2004.

[2] P. Buhler and J. M. Vidal, "Semantic Web Services as Agent Behaviors," in Agentcities: Challenges in Open Agent Environments, LNCS/LNAI, B. Burg, J. Dale, et al., Eds. Berlin: Springer-Verlag, 2003.

[3] R. T. Fielding, " Architectural Styles and the Design of Network-based Software Architectures", Ph. D. thesis, University of California, Irvine, 2000.

[4] M. Baker. "Ian Foster on Recent Changes in the Grid Community", IEEE Distributed Systems Online, 5(2), February 2004.

[5] T. D. Hodes, R. H. Katz, E. Servan-Schreiber, L. Rowe, "Composable Ad-hoc Mobile Services for Universal Interaction", Proceedings of the 3rd ACM International Conference on Mobile Computing and Networking, pp. 1-12, 1997.

[6] IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html

[7] IETF RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1", http://www.ietf.org/rfc/rfc2616.txt

[8] IETF RFC 2806, "URLs for Telephone Calls", http://www.ietf.org/rfc/rfc2806.txt.

[9] IETF draft RFC, "SMS URI Scheme", http://www.ietf.org/internet-drafts/draft-wilde-sms-uri-11.txt.

[10] M. Jones, G. Buchanan, G. Marsden, M. Pazzani, Improving Mobile Internet Usability. Proceedings WWW'10, Hong Kong. 2001.

[11] J. Kangasharju, Sasu Tarkoma, Kimmo Raatikainen, " Comparing SOAP Performance for Various Encodings, Protocols, and Connections", LNCS Lecture Notes in Computer Science, Volume 2775 / 2003, pp. 397 - 406, 2003.

[12] Christian Lindholm, Turkka Keinonen, "Mobile Usability: How Nokia Changed the Face of the Mobile Phone", McGraw-Hill Professional, 2003.

[13] G. E. Moore, Cramming more components onto integrated circuits, Electronics Magazine, 38(8), pp. 114-117, April 1965.

[14] M. Musolesi, C. Mascolo, S. Hailes. Adapting asynchronous messaging middleware to ad hoc networking. In Proceedings of the 2nd Workshop on Middleware For Pervasive and Ad-Hoc Computing, pp. 121-126, ACM Press, New York, NY, 2004.

[15] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Software Engineering Notes 17(4), pp. 40-52, October 1992.

[16] M. Ramsey and J. Nielsen. The WAP Usability Report. Neilsen Norman Group, 2000.