MobileDate or How to Engineer Ad-Hoc OSA/Parlay Applications

An advanced OSA/Parlay application, called MobileDate, is presented. MobileDate is identified as an ad-hoc application, in which no application-specific code is run on mobile nodes (users’ terminals). A general structure of ad-hoc applications implemented with the OSA/Parlay technology is described. It is observed that a great variety of such applications can be built by adding an application-specific logic module to a number of generic building blocks. This leads to a proposal for a middleware facilitating quick development of OSA/Parlay-based ad-hoc applications. The middleware resides on top of the OSA/Parlay APIs. Additionally, in order to enhance user interaction facilities, the middleware takes advantage of the WAP technology.

1. Introduction

OSA/Parlay [1] is an emerging technology that opens up the functionality of a telecommunications network to the “regular programmer,” thus enabling rapid creation of telecommunications services [2]. The functionality is exposed through a set of OSA/Parlay APIs (Application Programming Interfaces). Any of the major programming languages (e.g., Java) can be used. An OSA/Parlay application (service) runs on a central application server and uses the APIs by communicating with an OSA/Parlay gateway connected to the telecommunications network.

A class of ad-hoc applications has been defined recently [3]. Ad-hoc applications are distributed and run on mobile nodes (user’s terminals, e.g., PDAs or mobile phones) on behalf of their users. The users configure an ad-hoc application by entering their interests (preferences). When two mobile nodes running the application happen to be collocated (close to one another), the application layers communicate in a peer-to-peer fashion. They determine if the preferences match, and if so, the respective users are notified and can take an action (like meeting the other, “compatible” user). The concept of an ad-hoc application is expressed purely in terms of the application layer. It is independent of the underlying network technology, which can be either ad-hoc or infrastructure-based.

This paper presents MobileDate [4], an advanced OSA/Parlay application developed by Mobile and Embedded Applications Group [5]. MobileDate is identified as a kind of ad-hoc application, in which no application-specific code is run on mobile nodes; the application is run on an OSA/Parlay application server. Strictly speaking, according to the definition, such an application is not ad-hoc. It does not meet the peer-to-peer requirement, which claims that “pieces of software forming an ad-hoc application have to interact directly, without using a central server” [3]. This is simply because there are no distributed pieces of software. It is argued, however, that it makes sense to drop this implementation-oriented requirement – what really matters is the functionality. If so, MobileDate and similar applications, whose functionality is based on mobility, collocation, and matching preferences, are indeed ad-hoc.
In this paper a general structure of ad-hoc applications implemented with the OSA/Parlay technology is described. It is observed that a great variety of such applications can be built by adding an application-specific logic module to a number of generic building blocks. This leads to a proposal for a middleware facilitating quick development of OSA/Parlay-based ad-hoc applications (such as MobileDate). The middleware resides on top of the OSA/Parlay APIs. Additionally, in order to enhance user interaction facilities, the middleware takes advantage of the WAP technology.

This paper is organized as follows. The MobileDate functionality is presented in Section 2. The definition of ad-hoc applications, as found in [3], along with some comments, is given in Section 3. A general structure of ad-hoc applications developed with the OSA/Parlay technology is provided in Section 4. Section 5 includes the middleware proposal. Some suggestions on how to use WAP to enhance user interaction are presented in Section 6. The paper is concluded in Section 7.

2. MobileDate

MobileDate is an advanced telecommunications service that can be introduced in location-capable networks (i.e., in today’s cellular networks). The main goal of the MobileDate service is to enable making new contacts between people of matching preferences. Two service users are compatible when they mutually fulfill their specified preferences. The service is location-based. When two compatible users happen to be close to each other, the service takes an action depending on settings selected by the users (Fig. 1). One possible action is to make a third-party call connecting them. Another action is to notify the both users with an SMS that a compatible party is nearby. Either of them can then call the other one or send the other one an SMS.

The service is anonymous. The users do not know real telephone numbers of each other. Every user has her own unique user ID. The notification SMS generated by the service includes the user ID of the peer user. The user sends an SMS or calls another one using the peer user’s ID.

Two interfaces to manage the service are available: one is based on SMS messages and the other one on a Web interface.

There are two actors: a user and service administrator. The user has both interfaces (SMS and web). The user can register to the service, provide a description of herself, specify her preferences (sex, age, height, hair color etc.), manage her service settings, and see whole the history of connections generated by the service. Users can modify their preferences at any moment. Users can also change the actions taken by the service: they can specify if they prefer SMS or voice communication. Moreover, they can switch off the service in virtually any time. The so-called “blacklist” functionality allows ignoring parties that match the preferences but are not welcome.

![Fig. 1. Proximity-triggered events in MobileDate](image-url)
The administrator, in turn, uses the web interface only. He can see traffic generated by the service and related to a specific user.

MobileDate has been developed using the ATS OSA/Parlay emulator from Open APIs Solutions [6].

3. Ad-Hoc Applications

Consider a distributed application running on a number of autonomous nodes. As defined in [3], it is an ad-hoc application, if it meets the following three requirements.

1. The nodes are mobile.
2. The nodes (more specifically, their application layer software) interact in a peer-to-peer fashion (i.e., a central server is not visible to the programmer).
3. The interaction can occur if and only if the nodes are collocated (physically close to one another). The node-to-node (application layer) interaction can thus easily result in a real user-to-user encounter.

An ad-hoc application does not have to be run on top of an ad-hoc network. In fact, the above definition makes no assumptions as to the underlying network technology. The listed requirements apply either to the functionality of the application (the first and third one) or to the programmer’s view of node-to-node communications (the second one). The underlying network can just as well be ad-hoc (e.g., WLAN in the ad-hoc mode) as infrastructure-based (e.g., GSM/GPRS). In the latter case, the nodes could be Java-enabled phones running J2ME programs (MIDlets).

In particular, the second (peer-to-peer) requirement does not preclude a central server supporting an ad-hoc application; it only says that the programmer should not see a central server. The APIs used by the programmer should expose facilities enabling logically direct (peer-to-peer) communications between nodes. For some underlying network technologies, one cannot avoid using a central server. For example, the Mobile Information Device Profile (MIDP) of J2ME does not allow direct phone-to-phone communications. One can make an HTTP, socket (TCP), or datagram (UDP) connection between a phone and an Internet host (server). However, a middleware layer can be built on top of such client-server APIs. The middleware can hide the existence of the central server from the ad-hoc application programmer. The middleware API should be based on the peer-to-peer communications model.

As for the third (collocation) requirement, the collocated nodes, upon detecting one another, exchange some application layer interest (presumably the users’ preferences). Further communications between the nodes (and possibly the encounter of the users) takes place if the preferences match.

An example of an ad-hoc application is Ubiquitous Flea Market, described in [3]. Users can act as buyers or sellers (or both). The users express their interests by specifying items they want to buy or sell. If two compatible users happen to be collocated, they are notified and can meet and conclude a transaction.

Location is a key element of the so-called context (the current status of the user and her environment). Further, the preferences of ad-hoc application users can include statements about a desired state of context. Besides location, the context may include date and time (“notify me in weekend evenings only”), weather (“only if it’s hot and not rainy”), day-or-night information (“only after dark”), and the like. Thus an ad-hoc application is inherently context-aware. Obviously, the underlying platform must be able to acquire the context information from the environment and deliver it to the application.
4. Implementing Ad-Hoc Applications with OSA/Parlay

As noted, there are two types of requirements in the definition of the ad-hoc application. One type deals with the application’s functionality (the mobility and collocation requirement), while the other one has to do with implementation (the peer-to-peer requirement). One can argue that the implementation is not nearly as important as the functionality. Hence, it makes sense to drop the implementation-related requirement from the definition. In this paper we adopt such a reduced definition of ad-hoc applications. The reduced definition includes functionality related requirements only (i.e., mobility and actions triggered by collocation and common interest). If so, we claim that one can develop ad-hoc applications using the OSA/Parlay technology.

As an argument, consider the MobileDate application. It is implemented as a pure OSA/Parlay application. All the software is run on an OSA/Parlay application server; none application-specific software is run on the mobile phone (and so the second requirement of the original definition cannot be met). On the other hand, the mentioned above Ubiquitous Flea Market could be implemented very differently (e.g., on top of a true ad-hoc network). The two different implementations are depicted in Fig. 3. Yet, in spite of the implementation differences, the two applications are functionally extremely similar.

There may be a great variety of OSA/Parlay-based ad-hoc applications, each with its unique preferences and behavior specifics. MobileDate is a dating service. Another example would be matching people who share the same interests (e.g., collect antique coins). Ubiquitous Flea Market could be developed with OSA/Parlay as well. It is easy to see that all of them share the same basic structure. A generic architecture of the OSA/Parlay-based ad-hoc application is shown in Fig. 4.
The system consists of a number of elements. The WWW/WAP interface allows users to set their preferences (what they want) and capabilities (what they have). Apart from preferences and capabilities, users are able to suggest how the service should act when it detects a collocation (user settings). Users can change settings, preferences, and capabilities to configure the service. The information is stored in databases.

The context provisioning modules supply the application with information about the current status of service users and the environment. The key context element to be obtained from the telecommunications user context module is location. The module retrieves context information with OSA/Parlay APIs. As indicated, much more context information could potentially be processed by the service (to make it more attractive). Therefore, another context module is added to the architecture.

The matchmaker continuously processes the preferences, capabilities, and context information to find collocated compatible users. If such users are found, an event is generated.

The network control module allows the application to take action. For example, a call between the compatible users can be set up or an SMS can be sent to them. OSA/Parlay APIs are used by the module.

Finally, the service logic is the module that implements the service functionality. It acquires user settings and configures the matchmaker. In response to events generated by the matchmaker, the service logic uses the network control module to take an appropriate action.

5. Middleware for OSA/Parlay-Based Ad-Hoc Applications

As indicated in Fig. 4., OSA/Parlay-based ad-hoc applications share a number of building blocks that can be made generic and reusable. In fact, only the service logic module is application-specific. As a general approach, once some components have been identified to be common to a whole class of applications, they can be grouped into a middleware layer. The existence of such a reusable middleware can greatly speed up the development of future applications. Apart from reusability, the middleware can expose APIs that are significantly easier to understand and use than
those of the underlying layer. This is particularly relevant in the case of OSA/Parlay; while OSA/Parlay APIs were designed for the “regular programmer,” they are by no means straightforward. Based on the above, we propose an OSA/Parlay based middleware for ad-hoc applications. The middleware platform is schematically presented in Fig. 5.

![Middleware diagram](image)

**Fig. 5. Ad-hoc application middleware**

Functionality of the service is implemented by the application shown at the top. The middleware should supply the application with the following APIs:

- **Configuration API.** The API makes it possible to (a) describe settings, preferences, and capabilities, (b) specify user interfaces for entering them, and (c) set up databases for storing the information. Settings, capabilities, and preferences must be described in a way which is at the same time simple, formal (to enable automated processing), and versatile (to support a broad range of services). Once the user has configured the service, the API provides access to data on how the application should act when a match is found (settings).

- **Matchmaking API.** The application must be able to configure the matchmaker and to specify callback routines that the matchmaker will call when events occur.

- **Phone Call API.** This functionality gives the application means to make third party calls and reroute user-started calls to other service users.

- **User Interaction API.** This API gives the application means to interact with the user in a simple way. It will utilize a simple user interaction mechanisms (e.g., displaying textual information), messaging (SMS/MMS), and more advanced WAP-based, graphical interface.

The middleware should take advantage of the following OSA/Parlay service capability features (i.e., functional building blocks of the OSA/Parlay APIs):

- **Call Control SCF** to make third-party voice calls. In some cases more than two participants will use the service; therefore, the Multiparty Call Control version should be applied.

- **Mobility SCF** to localize users (interactively or using so-called triggered localization) and getting informed about the user status (reachable, unreachable, busy).

- **Messaging SCF** to send and receive the SMS/MMS messages.

- **User Interaction SCF** to make the service more interactive. The functionality of this SCF covers collecting numerical information, displaying short messages on users’ terminals and playing announcements. Since the SCF was designed to work in most networks (including PSTN), it is limited to the facilities just mentioned.
6. WAP-based user interface

As indicated above, user interaction capabilities of the OSA/Parlay APIs (more specifically, the User Interaction SCF) are quite limited. For example, with UI SCF, it is not possible to present the user of a mobile node a form to be filled out. Forms are a convenient way to specify preferences and so are a desirable feature of ad-hoc applications. Our assumption is that no application-specific software runs on the nodes. This precludes J2ME based solutions (MIDlets). Therefore, as shown in Fig. 5, our proposed middleware takes advantage of the WAP technology. The goal is to offer applications an easy to use API to generate and manage a graphical user interface (e.g., forms) on a mobile node. The middleware acts a translator between service requests issued by the application (API calls) and the WML pages delivered by a WAP server. The WML content [7] is generated and inspected by the middleware in response to the application’s activity (API calls) and the user’s activity. A generic WAP server and WAP browser in the terminal are used for transferring information.

A key problem with this approach is how to start an exchange between the user’s terminal and the server. The case of user-initiated exchange is typical. For example, if the user wants to update her preferences, she can simply use WAP to get a WML page containing a respective form. An application-initiated exchange can be set up in at least two different ways. The preferred one is to use the WAP Push technology. A middleware API call (e.g., to display a form on the user’s terminal) causes a WAP Push SMS to be sent. The other way is to send a regular SMS containing a WAP link instead of a WAP Push SMS (only if link recognition in SMS messages is supported by the mobile phone). In either case, the user downloads the WML page.

The middleware API call that was translated into the initial WML page can be either a non-blocking or a blocking one. If it is non-blocking, and further interaction with the user is expected, the <ontimer> and <refresh> WML tags can be used to make the WML browser to periodically download updated user interface content. This method, however, could be quite expensive. A more elegant approach is to have a blocking API call – the initial WML page is displayed, and the application is blocked until the user enters some data (e.g., fills out a form). Once the user submits the form, a GET method is used to send the data to the server. The data is returned by the middleware to the application (as the return value from the blocking call). The application can issue another API blocking call, which will produce a new WML page. The new page will be sent to the terminal in response to the GET method. The blocking call method does not incur any overhead beyond what is absolutely necessary.

7. Conclusions

MobileDate, a complex OSA/Parlay application, has been presented. A definition of ad-hoc applications, which does not imply ad-hoc networking technology, has been quoted [3]. It has been argued that only the functionality-oriented requirements should be preserved in the definition; the implementation-oriented requirement could be dropped. If so, ad-hoc applications can be developed using the OSA/Parlay technology. In fact, MobileDate is one of them.

It has been observed that such applications can be developed by adding an application logic module to a number of generic building blocks. A middleware layer for OSA/Parlay-based ad-hoc applications has been proposed. The purpose of such a middleware is twofold: (a) to make the building blocks reusable and (b) to hide complexity of the OSA/Parlay APIs. The middleware could greatly simplify the development of OSA/Parlay-based ad-hoc applications.

References
2. M. Rój, J. Domaszewicz, Tworzenie Usługi Telekomunikacyjnej Wykorzystującej Parlay/OSA API, KST, Bydgoszcz, 2002
3. B. Garbinato, P. Rupp, *From Ad Hoc Networks to Ad Hoc Applications*, 7th International Conference on Telecommunications (ConTel), Zagreb, Croatia, June 2003