IT University
of Copenhagen

The Context-Dependent Mobile Communication Platform

A Technical Report at the ITU

Thomas Bodin
Academic Officer
MSc in IT, Multimedia Technology

IT University Technical Report Series

TR-2006-TR 81

ISSN 1600–6100 **January 2006**

ISSN 1600–6100

ISBN 87-7949-120-0

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone:	+45 7218 5000
Telefax:	+45 7218 5001
Web	`www.itu.dk`

Abstract

Context-aware applications for guiding users exist in several versions, mostly using visual interaction. However, users should have eyes and hands free for more important tasks like navigating and observing their surroundings. This report describes the development of a piece of middleware, the context-dependent mobile communication platform, that facilitates easy construction of LBSs using speech interaction by handling much of the conversation logistics internally in the platform itself. The platform is based on both positioning and speech technologies, supporting dynamically generated system responses which can be provided to the user in a push or pull based manner.

In order to demonstrate the platform usability, an LBS has been implemented taking the main concepts of the DELCA (dis-embodied location-specific conversational agents) project. In short, the DELCA project presents a collection of *conversational agents* capable of generating synthesized speech by using existing speech technologies. Based on the positioning technology, these *location-specific* conversational agents will be able to make conversation with a registered user according to his/her location.

Acknowledgements

This report has been made with great assistance from Lars Birkedal (Head of LaCoMoCo at the IT University of Copenhagen), Arne John Glenstrup (Assistant Professor) and Henning Niss (Assistant Professor), who have provided valuable technical and textual feedback. Furthermore, I want to thank the people from the DELCA group for the meetings, where we have had the opportunity to present and discuss relevant problem areas. These meetings have influenced and inspired the platform design as well as the implementation. For relevant linguistic comments towards the report, I want to thank Elisabeth Bodin.

Contents

Chapter 1

Introduction

In this report we discuss and demonstrate how positioning and speech technologies can be integrated in a generic, extensible and scalable platform to support some of the basic features of location-based services (LBSs)[1]. Moreover, it must be examined how well hands-free and eyes-free communication can be supported on handheld devices, leading to a more user-friendly design compared to the ordinary graphical user interface (GUI) normally presented by LBSs today. Hands-free and eyes-free communication is supported by an auditive user interface (AUI) in which the user receives system responses represented by synthesized speech through a head set. The headset must also include a microphone, enabling the user to respond by using speech recognition. In both hands-free and eyes-free scenarios, the user does not have to focus on a screen in order to communicate. Obviously, this will be a big advantage in mobile applications, as the user can communicate without being interrupted in ongoing business (e.g. driving, walking etc.). In order to support these mobile applications the platform must be able to operate in a wireless LAN, thus it will be relevant to discuss, how the user is provided with a sense of continuity in the flow of information. As the synthesized speech is downloaded on the fly using wireless transmission, there is a potential risk of delays and disruptions. The attempt is to circumvent these problems firstly by caching audio on the handheld device (should the same sound bite be needed later) and secondly by keeping a small library of fixed pause fillers in the handheld device. Such pause fillers include coughing sounds, short excuses like “let me see...” that can be played to the user while waiting for a download to complete, thus keeping the user in the flow of conversation.

The above mentioned platform features can be specified in some general requirement specifications. Based on these requirement specifications, a platform prototype has been designed and implemented at the IT University of Copenhagen built on existing positioning and speech technologies.

1. The platform must support some of the basic features of LBSs.
2. The platform must be generic, extensible and scalable.
3. Hands-free and eyes-free communication on handheld devices can be supported based on an AUI.

[1] In short, an LBS provides the mobile device, the mobile application, and the mobile user with location information about themselves or other devices, applications, and users [2].

4. In order to support mobile applications, the platform must be able to operate in a wireless LAN.

Basically, the platform consists of two layers representing core technologies (positioning and speech technologies) as well as system components. Each of the system components satisfies a well-defined interface specification, thus each component can be tested, improved and/or replaced without effecting the surrounding system.

The platform prototype will be demonstrated through an LBS taking some of the basic principles of the DELCA (dis-embodied location-specific conversational agents) project[2] that has been started as a working group at the IT University of Copenhagen, where both researchers and students have had the opportunity to come up with new ideas relevant to the primary aim of the project. This primary aim has been to establish certain *conversational agents*, capable of communicating and offering services depending on the location of a certain user. Beside of being *location-specific* they are also *disembodied*, as they will not be represented by any physical form in the user interface. Therefore, they will only be able to use auditory communication such as speech, ambient sounds and music. Since these conversational agents have no physical appearance, they are often referred to as *ghosts*. Obviously, the platform requirement specifications nicely suit the basic principles of the DELCA project, thus it will be used as the general example in Chapter 4.

1.1 Motivation

By the increasing use of handheld devices and the possibility to compute context by sensor technologies, systems are moving more and more in the direction of context-awareness. While graphical user interfaces can be useful for stationary tasks (like office work, and to convey inherently graphical information like photographs, maps and graphs), audio based interfaces are to be preferred for performing tasks on the move, in cases where information is not intrinsically graphical, and when the amount of information is hard to fit into a small visual space. Consider a human guide walking a visitor around a building; the visitor would certainly be puzzled if this was done by drawing arrows on a map. Instead, the guide is supposed to tell things like “go this way”, “notice the view from the window” and “please mind the step”. In this case, audio does not require the use of hands and eyes, leaving them free for more important tasks when on the move, like navigation, opening doors etc.

The combination of speech and positioning technology might in many ways change the way people interact with computers relating to the research of pervasive and ubiquitous computing[3], and the technical experiences (derived by the platform prototype) are supposed to make a contribution to this research.

[2]The LaCoMoCo (Laboratory for Context-dependent Mobile Communication) at the IT University of Copenhagen is meant to initiate projects relating to ”context-dependent mobile communication” including the DELCA project.

[3]Quoted from Mark Weiser: ”Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user” [11].

1.2 Problem Statement

Based on existing positioning and speech technologies, a platform prototype will be designed and implemented in order to support the development of LBSs focusing on AUIs as well as on obtaining of meaningful conversations. Technical challenges involved in designing such a platform include how to perform well in the face of limited resources (memory, CPU speed, power) on handheld devices, and also how to cope with wireless transmission that has limited/varying bandwidth, latency and temporary transmission disruptions. Even given perfect wireless transmission, one must be ready to handle imperfect positioning information. Finally, the platform must be constructed in a modular way to make it generic, extensible and scalable.

An LBS (based on the basic principles of the DELCA project) will be implemented in order to provide experimental data used to evaluate the current platform prototype.

1.3 Report Guide

This section serves as a brief guide to the report covering the following 5 chapters:

Chapter 1, Introduction: Some of the main concepts about the platform and the applied technologies are being introduced. Furthermore, the chapter presents the problem statement in detail as a report foundation.

Chapter 2, System Design: The platform design constituted by system components is described as well as the currently used technologies.

Chapter 3, System Implementation: The implementation of a generic platform prototype is covered based on the system components as well as the applied positioning and speech technologies.

Chapter 4, Platform Demonstration: A demonstration of the platform prototype usability based on the basic principles of the DELCA project.

Chapter 5, Conclusion and Perspectives: A conclusion based on the experimental results in chapter 4 followed by the perspectives of future platform developments.

Chapter 2

System Design

In this chapter the primary aim is to integrate already existing positioning [6, 3] and speech [1] technologies into one generic platform. As the platform is supposed to support LBSs in different environments, it must be able to handle different kinds of positioning technologies in order to obtain dynamic update of location data from the connected handheld devices. The associated problem areas regarding reliability, latency and location representation will not be fully explained in this report, as they comprehend a full study of location models [8]. Still, they will be referred to briefly, as they are representing important properties regarding the platform design. Also, the platform may support LBSs based on speech communication leading to more user-friendly interfaces. This speech communication must be structured as small conversational patterns replacing the more common non-conversational commands[1]. In other words, the communication must have a conversational[2] quality.

The supported LBSs mainly apply to handheld devices, thus important questions regarding bandwidth and latency in a wireless environment must be brought to attention. Therefore, a given LBS must be prepared to handle unfavourable conditions, whenever these entities are either limited or varying - or both in same time. Also, some precautions must be taken regarding the limited resources on handheld devices to avoid demanding computations on the client-side.

The design considerations (cf. requirement specifications in Chapter 1) are listed below and will be discussed in connection with the system design leading to the current implementation of a platform prototype referred to as the Context-dependent Mobile Communication (CMC) platform.

1. According to the applied positioning technology, there will occur some erroneous location data, which will effect the performance of an LBS. Therefore the CMC platform may implement filtering algorithms to ensure a higher probability of correct location data. Also, one must assume some delay concerning the update of location data (cf. latency), which might be solved by prediction algorithms based on historical data.

2. The platform must be designed in a generic modular way, so that it easily can be maintained and expanded. Furthermore, it is assumed to work as an application programmers interface (API) designed to assist system programmers in developing new LBSs.

[1]The non-conversational commands are characterized by a series of mutually independent speech commands, contrary to the conversational patterns representing a series of interrelated sentences.

[2]In this context, a conversation is defined as a finite/infinite series of interrelated sentences exchanged between the user and system.

3. To fully obtain a hands-free and eyes-free communication (according to handheld devices) based exclusively on the auditive sense, it is necessary to apply both speech synthesis and recognition. Using this approach, the user will be able to communicate by sending (cf. speech recognition) and receiving (cf. speech synthesis) commands without the normal use of keyboard and screen. Moreover, this speech communication must have a conversational quality based on conversational patterns.

4. Limitited and varying bandwidths in a wireless environment will occasionally lead to slow data transfer rates (cf. latency) and possible loss of data packages. As the client-side of the communication mainly apply to handheld devices with limited resources, the design of a thin client architecture must be intended. This will require a lot of data exchange, as the demanding computations must be carried out at the server-side. Thus, the CMC platform must take relevant precautions regarding the design of a distributed system and the assigned application protocol between client and server.

2.1 Platform Design

Basically, the CMC platform (cf. figure 2.1) works as a middleware layer between the LBSs and the applied technologies. Seen from an LBS perspective, the CMC platform works as a *black box*, hiding the concrete implementation behind system component interfaces (SCIs), giving the usual benefits of easy modular update of implementations. Similarly, the system components (SCs) are also decoupled from the concrete implementation in the underlying technology components (TCs) by specifying technology component interfaces (TCIs). Currently, the Ekahau Positioning Engine 3.1 and AT&T Natural Voices constitute the positioning and speech technology in the CMC platform, though these can easily be replaced without affecting the existing SCs.
The communication flow in the CMC platform divides into event based (EB) and query based (QB) information corresponding with the *push* or *pull* service of an LBS. The EB information relates to the publish-subscribe paradigm, in which events are published to a list of subscribers [5]. This correponds to the push service defined by the LBS, where the system delivers information (events) without an LBS request. According to the publish-subscribe paradigm, the LBS must be on the list of subscribers, if this is supposed to work. On the other hand, the QB information uses a request-reply protocol matching requests to replies [5]. Basically, it ensures that only requests configured in the application protocol will be replied. In the LBS this corresponds to the pull service, where the handheld device receives a reply according to a specific query (request).

In the following, the current functionality of the CMC platform will be further explained, including the four SCs presently available: speech, conversation, positioning and distribution.

2.1.1 Speech

The main functionality of the speech SC will be to generate synthesized speech based on text input. The synthesized speech should be returned in a specific audio format as a byte array. Also, it should be possible to either state audio properties (such as sample rate and bit quantitation in order to reduce the file size) or to state the file size maximum with default audio properties. If the text length (cf. number of characters) exceeds a specific maximum limitation,

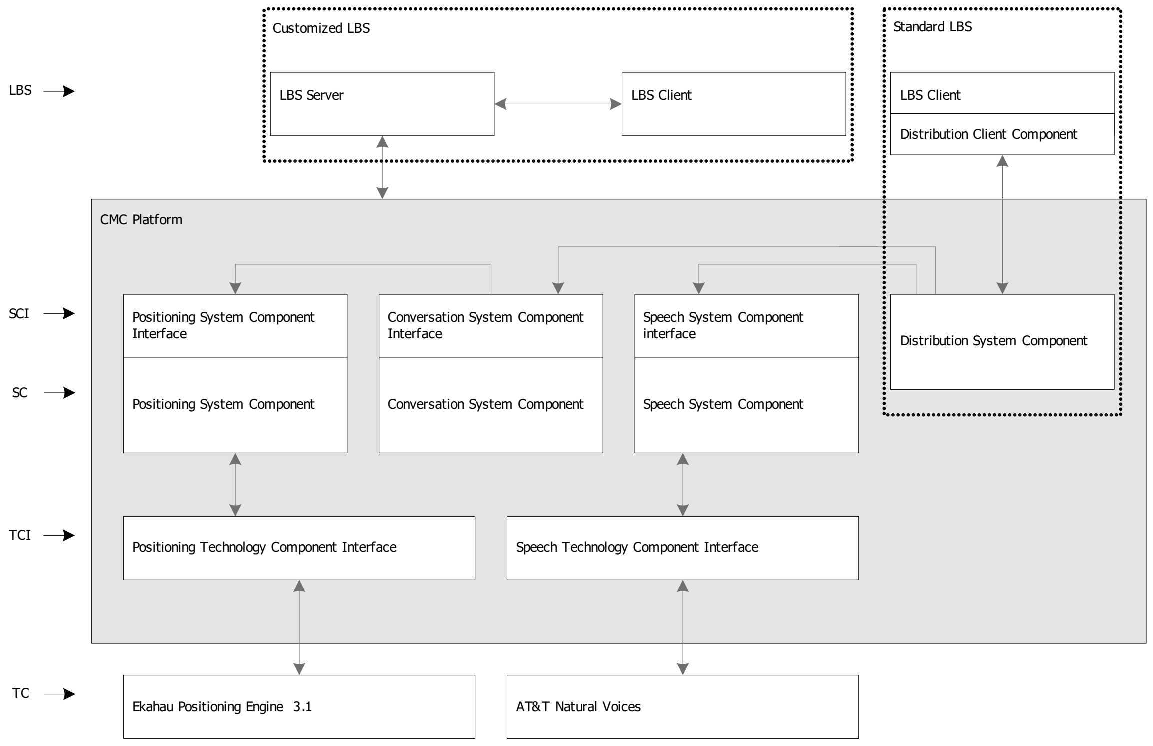
Customized LBS
Standard LBS
LBS
LBS Server
LBS Client
LBS Client
Distribution Client Component
CMC Platform
SCI
Positioning System Component Interface
Conversation System Component Interface
Speech System Component interface
Distribution System Component
SC
Positioning System Component
Conversation System Component
Speech System Component
TCI
Positioning Technology Component Interface
Speech Technology Component Interface
TC
Ekahau Positioning Engine 3.1
AT&T Natural Voices

Figure 2.1: The current CMC platform with the SCs (positioning, conversation, speech and conversation) and their corresponding SCIs, as well as the TCIs (positioning and speech) and TCs (Ekahau Positioning Engine 3.1 and AT&T Natural Voices).

the procedure will terminate in order to restrict the size of rendered data per client request, and in turn the audio download time.

Please note that the speech SC provides only synthesized speech and does not handle speech recognition, as this process would introduce some serious problems regarding performance. Indeed, it is problematic to decide when a certain client application is supposed to transmit audio signals to the speech recognizer. One solution would be to put a certain threshold on the audio recording at the client-side in order to discern background noises from input speech. However, one cannot avoid that irrelevant audio signals (e.g. cough, sneeze, yawn etc.) will be recorded. Thus, redundant data will be transmitted to the speech recognizer, where it will be rejected as unrecognizable speech after having been processed. This redundancy will increase the network traffic, as all audio signals applying to a certain threshold will be accepted and transmitted. Therefore, the speech SC does not handle speech recognition, as it will both result in an undefined amount of redundant data as well as an unnecessary overhead towards the processing of audio signals by the speech recognizer. On the contrary, if the speech recognition is carried out on the client-side the result will be a reduced network activity, as no redundant data will be transmitted to the server-side. Furthermore, the data packages will decrease in size, when a string representation of the recognized speech can be transmitted instead of a complete audio signal. Although this solution solves some important problems, it introduces another problem concerning limited resources on handheld devices.

The Speech SCI

The speech SCI declares a collection of QB methods in order to open and close a connection to the proper speech engine, and to obtain the synthesized speech (cf. figure 2.2). The `openConnection` method may throw an exception according to network or engine problems caused by e.g. wrong `hostAddress`, engine breakdown etc. Basically, the obtaining of synthesized speech is done by calling one of the two `getSynthesizedSpeech` methods (see Section 3.1.1) that will throw an exception, if an unexpected error occurs in the ongoing processing of synthesized speech (e.g. the text length exceeds a specific maximum limitation).

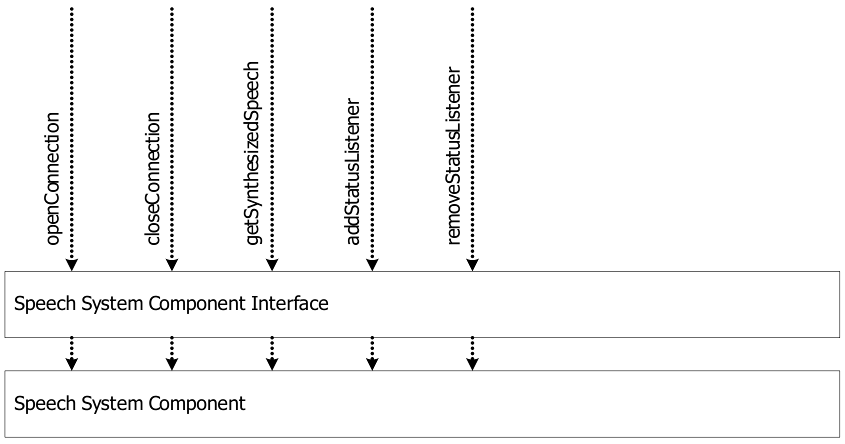

Figure 2.2: The speech SCI and the declared methods.

In order to perform status notification of the processing of synthesized speech, an EB method is provided as well. This functionality is provided by an observer [7] pattern available through the `addStatusListener` method.

2.1.2 Conversation

Basically, the conversation SC contains all the logistics of the conversation between system and user. In the current definition of a conversation a differentiation has been made between static and dynamic conversations. In a static conversation all possible conversations have been written to the system beforehand. If the user is not supposed to have long and intelligent conversations with the system, this might be a sufficient solution. Otherwise, the system must utilize artificial intelligent chat robots that can reflect on input sentences based on a collection of rules[3]. Still, this will not be truly dynamic, as any AI will have its limitations matching the collection of rules. As figure 2.3 shows, a typical conversation can be described by a tree structure. One

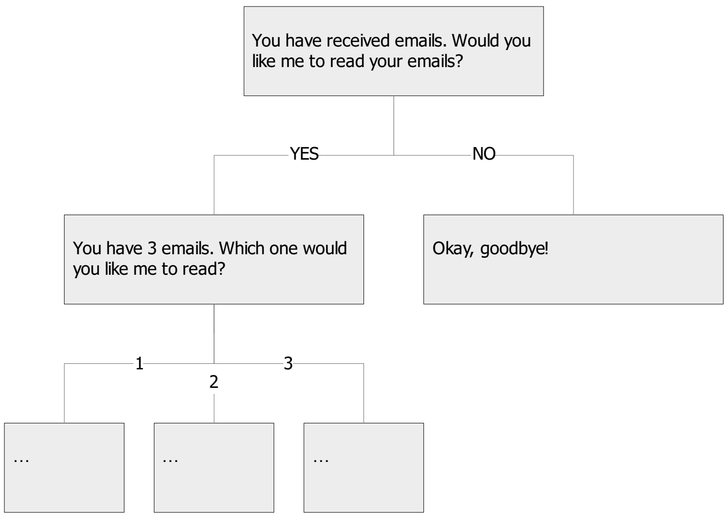
You have received emails. Would you like me to read your emails?
YES
NO
You have 3 emails. Which one would you like me to read?
Okay, goodbye!
1
2
3
...
...
...

Figure 2.3: The tree structure shows a simple example of a possible static conversational pattern, where the leaves are expressing the system response, whereas the branches are expressing the user response.

way of representing this tree structure is by an XML semi-structured document that can easily be written and expanded in order to configure the system, as shown in figure 2.4. Besides the main `conversation` tag, the structure of this XML document is very simple, as it declares only two tags according to the tree structure shown in figure 2.3. The `user` tag refers to the user request, and the `agent` tag refers to the system response corresponding to the leaves in the tree structure. The `agent` tag is supplied with attributes representing some specific system properties such as `file` (cf. applied audio file) and `text` (textual representation of the audio file). Furthermore, the `user` tag is supplied with the attribute `response` representing the pre-configured user responses.

The XML document can result in different information sequences depending on the `re-sponse` attribute in the `user` tag. When the XML document is processed, the information from the first `agent` element node will be extracted. Thus, the `file` attribute (according

[3]The Artificial Intelligence Mark-up Language (AIML) is an XML-compliant language used to program chat robots.

```
<conversation scenario="static">
  <agent file="mail.wav" text="You have received mails. Would you like me to read them for you?">
    <user response="Yes">
      <agent file="mail_yes.wav" text="You have 3 mails. Which one would you like me to read?">
        <user response="1">
          <agent file="mail_1.wav" text="..."/>
        </user>
        <user response="2">
          <agent file="mail_2.wav" text="..."/>
        </user>
        <user response="3">
          <agent file="mail_3.wav" text="..."/>
        </user>
      </agent>
    </user>
    <user response="No">
      <agent file="mail_no.wav" text="Okay, goodbye!"/>
    </user>
  </agent>
</conversation>
```

Figure 2.4: The tree structure of figure 2.3 can be represented by an XML semi-structured document. This example shows an XML document (4C01.xml) for describing a static conversational pattern.

to figure 2.4) will be ”mail.wav” and the corresponding `text` attribute will be, ”You have received emails. Would you like me to read your emails?”. Depending on whether the `response` attribute (in the `user` tag) in this case is ”Yes” or ”No”, the element child node will be extracted accordingly.

Besides the `agent` and `user` tags, the XML document also includes a main `conversation` tag, where the `scenario` attribute refers to a specific type of convertsation scenario. This scenario can be either dynamic or static (cf. figure 2.6). In the former XML document (cf. figure 2.4), the `scenario` attribute refers to a static scenario indicating that this XML document is statically bound to a specific area. Thus, it will be triggered, whenever a user enters a related area, and the processing will depend on the user interaction. Likewise, all processing of the XML document will terminate, if the user leaves that specific area. On the other hand, the `scenario` attribute can refer to a dynamic scenario indicating that the processing of this XML document will depend on the user location (contrary to the user interaction) provided by the positioning SC (cf. Section 2.1.3). An XML document representing a dynamic scenario is illustrated in figure 2.5. In this example, the user should be directed from area `4Cc1` to `4Cc4`

```
<conversation scenario="dynamic">
  <agent file="4Cc1.wav" text="Please, walk down the long corridor.">
    <user response="4Cc2">
      <agent file="4Cc2.wav" text="Please, continue straight out.">
        <user response="4Cc3">
          <agent file="4Cc3.wav" text="Please, do continue down the long corridor.">
            <user response="4Cc4">
              <agent file="4Cc4.wav" text="The door leading to room 4C03 is on your right hand."/>
            </user>
          </agent>
        </user>
        <user response="4C01">
          <agent file="4C01.wav" text="You where supposed to continue down the long corridor."/>
        </user>
      </agent>
    </user>
  </agent>
</conversation>
```

Figure 2.5: Example XML document (4Cc6.xml) for describing a dynamic conversational pattern.

passing area `4Cc2` and `4Cc3`, and the user is supposed to move exactly according to the XML document structure to reach the destination target. This scenario can also be described by a path referring to a list of sorted areas.

`PATH = {4Cc1, 4Cc2, 4Cc3, 4Cc4}`.

The XML document could be made much more complex, if the user should be redirected, whenever he/she moves in a wrong direction. Also, the XML document could be generated dynamically based on computations of a shortest path algorithm. In that case, the areas will correspond to the nodes in a weighted graph, and the allowed directions from a certain node will be defined by the edges [4].

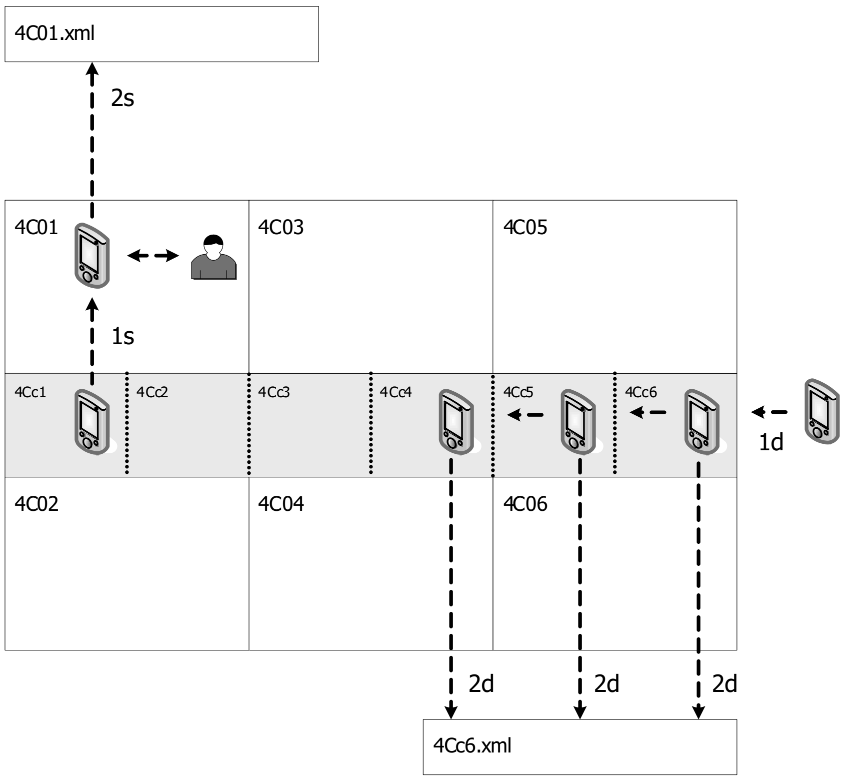

Figure 2.6: An office map illustrating static and dynamic scenarios. In the static scenario, a handheld device is registered in room 4C01 (1s) causing the static XML document, 4C01.xml, to be triggered. Subsequently, this XML document is processed based on the user response (2s). In the dynamic scenario, the dynamic XML document, 4Cc6.xml, is triggered when a handheld device is registered in corridor section 4Cc6 (1d). Subsequently, this XML document is processed based on the user location (2d) provided by the positioning engine.

The Conversation SCI

The conversation SCI provides a simple collection of QB methods in order to open and close a connection to a possible conversation engine, and to get and set user responses. The `openConnection` method may throw an exception according to network or engine problems caused by e.g. wrong `hostAddress`, engine breakdown etc. The two methods, `getResponse` and `setResponse`, will simply get and set the user response referring to an ongoing conversation, thus hiding the complexity of the conversation SC. Both methods will throw an exception, if the request cannot be computed due to an ongoing conversation.

Due to the EB method, `addConversationListener`, two types of notifications can be spawned. Firstly, a notification is spawned, whenever a new conversation has become available. This will happen, when the user enters a specific location, which will trigger a current conversational pattern. Secondly, the ongoing conversation will notify the observer, whenever a new system response has become available.

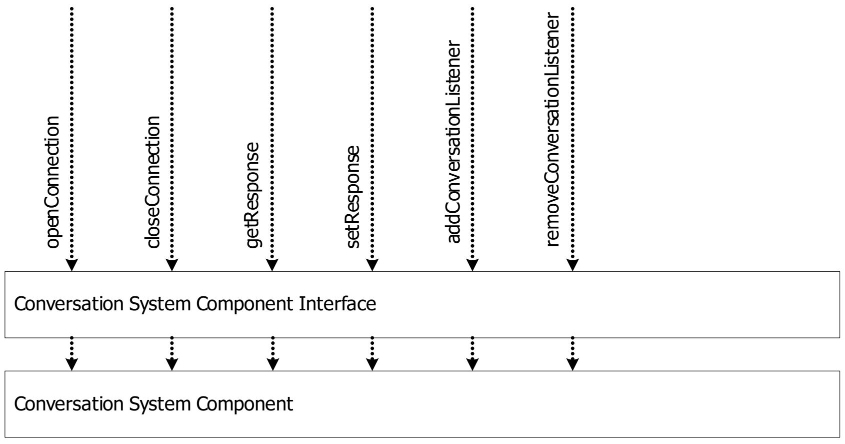
openConnection
closeConnection
getResponse
setResponse
addConversationListener
removeConversationListener
Conversation System Component Interface
Conversation System Component

Figure 2.7: The conversation SCI and the declared methods.

2.1.3 Positioning

The main functionality of the positioning SC is to decide the location of a specific handheld device or any kind of object capable of being tracked. This can be enabled by certain indoor/outdoor positioning technologies. A typical outdoor positioning technology is the Global Positioning System (GPS), where the handheld device receives a collection of signals sent from satellites. The knowledge about the location of each satellite and the time measurement of each signal is then used to compute the outdoor location of the certain mobile unit.

The indoor positioning technology must use another approach, since the satellite signals used in the GPS are prevented from entering by the construction materials of a building [10]. Instead, the indoor positioning technologies are based on wireless standards such as Bluetooth (IEEE 802.15.1) or WLAN (IEEE 802.11b). In the current implementation of the CMC platform, a WLAN based positioning technology has been used[4], but this technology could easily be replaced (according to the TCI) by another sensor technology such as Bluetooth. The WLAN based positioning technology uses access points (APs) described as small physical devices (antennas) sending out certain signal strengths. Based on a manually performed calibration model[5] in a building, the placement of a handheld device can be estimated accordingly. The handheld device sends a package of currently registered signal strengths from APs within range, and the positioning engine returns the analogous location.

The Positioning SCI

In order to make the location data accessible, the positioning SCI provides a collection of QB methods to open and close a connection to a positioning engine, and to receive information

[4]Currently, the IT University of Copenhagen uses the Ekahau Positioning Engine 3.1 (cf. Section 3.1)

[5]In the calibration model the signal strengths are measured for each calibration point in a grid covering the entire building [6].

about tracked devices. The `openConnection` method may throw an exception according to network or engine problems caused by e.g. wrong `hostAddress`, engine breakdown etc. The `getTrackedDevices` method will either return a list of all tracked handheld devices or just those registered in a specific area. Furthermore, the location of a tracked handheld device can be obtained by calling the method, `getLocation`. Accordingly, the method, `getLocations`, will return a list of locations based on a collection of device id's.

In order to provide location notification an EB method must be declared by the positioning SCI. The `addTrackingListener` method uses an observer pattern in order to notify the registered observers about new tracked locations [7]. This method ensures a continuous location data notification related to all connected handheld devices.

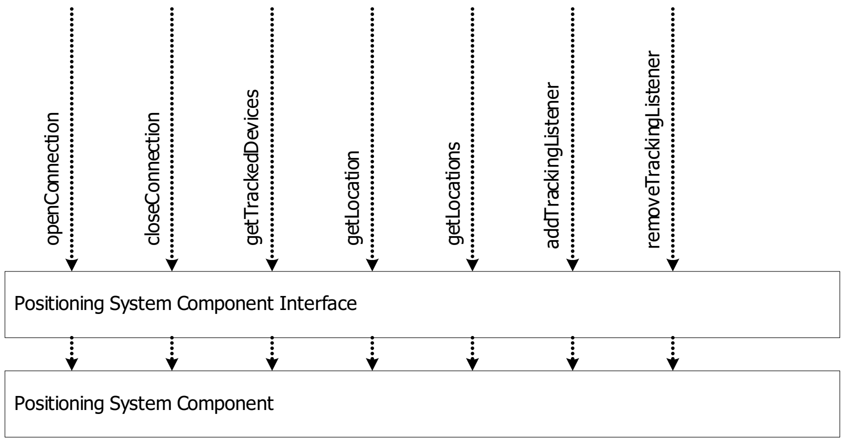

Figure 2.8: The positioning SCI and the declared methods.

2.1.4 Distribution.

The distribution SC simply provides a short cut for the LBS system programmer to incorporate an already designed distributed system (cf. figure 2.1). By importing the distribution CC on the handheld-device, the enclosed API can be used in order to obtain a standard functionality of the CMC platform[6].

Basically, the distribution SC must transmit synthesized speech in bytestreams as well as an array of pre-defined user responses extracted from a currently active conversational pattern. The LBS client can use this array (received by the distribution CC) either to define a speech grammar file (used by the speech recognizer to specify a specific set of recognizable words and/or sentences), or to create choices in an ordinary GUI. The distribution CC will transmit only a string representation of the user response, as the speech recognition is assumed to be processed by the LBS client.

Also, authentication must be provided in order to support a future security policy based on a certain device id (cf. MAC address, Bluetooth address etc.) referring to a specific LBS client

[6]As figure 2.1 shows, the system programmer can also make a customized distributed system, if the LBS has some special requirements that are not covered by the standard distribution SC/CC. In that case the LBS will consist of both a client-side (LBS client) running on the handheld device and a server-side (LBS server) using the CMC platform functionality.

profile. Thus, the application protocol between the distribution SC and CC will handle the following data formats in order to meet some standard requirements regarding authentication and data exchange:

1. In the authentication process the distribution CC will transmit a username, password and device id (string representation) based on a log-on procedure provided by the LBS client. A new session will only be initialized, if the distribution SC receives a username, password and device id that is legal according to an authentication database. Also, the authentication ensures that a certain username and password cannot occur twice at the same time, which would be problematic when updating the profile of a certain user.

2. When the synthesized speech has been produced, this must be transmitted as a bytestream to the distribution CC, along with a string representation and a string array consisting of pre-defined user responses (extracted from the current XML document). This information is forwarded by the distribution CC to the LBS client. When the LBS client responds with one of the pre-defined user responses, it will be transmitted by the distribution CC in a string format to the distribution SC on CMC platform.

Based on these steps, the application will be able to support scenarios such as the following: Firstly, the LBS client is supposed to perform a log-on procedure (cf. distribution CC) where username, password and device id are transmitted. In order to authenticate, the distribution SC will query an authentication database and transmit the result back to the LBS client. If the authentication is successful, a new session will be spawned by the distribution SC provided that the necessary SCs (speech, conversation and positioning) can be obtained. Whenever, the LBS client moves to a certain area, where an XML document (cf. conversational pattern) resides, a new conversation object will be spawned by the conversation SC. The conversation object will now process the current XML document starting with the first element node. Firstly, the attribute values of the first `agent` node will be processed leading to the transmission of a synthesized speech represented as a bytestream. According to the example below (cf. figure 2.9), the synthesized speech will be, ”Would you like to find Peter?” Secondly, the child element node of the former `agent` node is processed in order to transmit the possible `response` values of the `user` node. These values are used by the LBS client to either build a speech grammar file or to provide a GUI with possible choises.

```
<agent file="find_peter.wav" text="Would you like to find Peter?">
  <user response="Yes">
    <agent file="find_peter_yes.wav" text="He is in room 4A21"/>
  </user>
  <user response="No">
    <agent file="find_peter_no.wav" text="Okay, goodbye!"/>
  </user>
</agent>
```

Figure 2.9: An example showing a possible scenario involving a static XML document.

When the choice has been made by the LBS client, the user response is transmitted back to the distribution SC. Based on the XML document above, the user response will now be evaluated. When no more element nodes are available in the XML document, the last child node (cf. `agent`) of the element node will be processed, followed by a termination of this conversation.

This example does not handle the problems regarding varying bandwidth and latency in a wireless network, though the wireless property is crucial in the distribution process. A wireless network must be expected to be unstable leading to slow data transmissions that might effect the entire system performance, and there is a possible risk that an ongoing transmission will be interrupted, if an established socket connection fails. Evidently, these two problem areas must influence the design of the application protocol of the current client-server model:

1. In order to reduce the transmission of byte arrays, the distribution SC only sends a specific byte array, if this cannot be found in a cache on the handheld device. To make this look-up service possible, the distribution SC transmits a filename (cf. attribute `file` of the `agent` node) and awaits the distribution CC response. If the response is negative, the distribution SC will transmit the byte array. Accordingly, the distribution CC will cache the byte array under the specified filename.

2. If temporary transmission disruptions occurs between the distribution SC and CC, this must be handled by the LBS client as well. As a session object is created simultaneously with the socket connection, this same session object can be used when the socket connection is re-established. However, the session object should only exist in the virtual memory for a short time in order to distinguish between a network exception and a normal log-out procedure. The distribution CC must re-establish the socket connection behind the scene in order to avoid a new log-on procedure by the LBS client.

Due to usability in a real world situation, the LBS client is supposed to provide the user with some information or service based on a current location. If the transmission is too slow or the transmission is disrupted, the user might have left the location before the transmitted data is received by the handheld device. Thus, it can be measured how effective the network is at a given time in order to inform the user about the current progress. A certain time limit will be controlled by a timer started by the distribution CC whenever a byte array is expected. In the current LBS client the user receives a “bad excuse” delivered by some synthesized speech, pre-loaded into the handheld device if the time limit is exceeded. This simple solution does not solve the transmission latency in the network, but it makes the LBS client appear more user-friendly.

The Distribution CC

The distribution CC provides a simple collection of QB methods applying to the LBS system programmer. Basically, these methods will handle the logistic concerning opening a connection and transmitting data based on the application protocol between the distribution SC and CC. The `openConnection` method may throw an exception according to network or engine problems caused by e.g. wrong `hostAddress`, engine breakdown etc. The `transmitAuthentication` and `transmitResponse` methods are responsible of transmitting data from the distribution CC.

The two EB methods, `addAuthenticationListener` and `addDistributionListener`, respectively will handle the notification about authentication results as well as received data such as incomming byte streams.

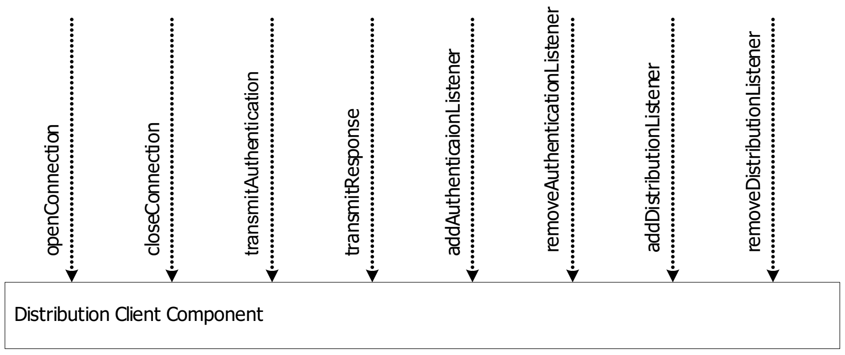

Figure 2.10: The distribution CC (client component) and the declared methods.

2.2 LBS Design

In order to demonstrate the usability of the CMC platform, an LBS client has been designed. This LBS client is closely related to the DELCA project [9], thus it will be demonstrated how conversational agents (ghosts) can be built directly on the CMC platform by using the standard functionality of the SCs described above. As the DELCA project [9] presents a comprehensive collection of conceptual ideas within the area of ubiquitous computing, only a very few can be realized by this first CMC platform prototype, where the purpose mainly has been to demonstrate the usability of the SCs described above and to show how the generic concept of the SCIs is utilized.

Basically, the current LBS client only handles a GUI regarding the user interaction, thus it consists merely of an authentication GUI with reserved fields for username, password and device id. When the authentication is performed, the GhostAuthentication object calls the `transmitAuthentication` method of the distribution CC and awaits the system response. If the authentication is successful, the `GhostDistribution` object is initiated, and a session (controlling the input and output streaming) between the distribution SC and CC is established.

Chapter 3

System Implementation

The requirements of the CMC platform are only partially implemented, as some of the suggested solutions are not trivial and can be seen as future developments. Thus, this chapter will present what currently has been implemented in the CMC platform prototype. In addition to, a description of the implementation issues attached to the speech, conversation and positioning SC, the distribution SC/CC will be thoroughly explained, as it introduces possible solutions (based on the main concepts of the CMC platform) on how to handle the limited/varying bandwidth, latency and temporary transmission disruptions as described in Section 2.1.4.

The demonstrated LBS client mainly consists of a GUI implementation, as it uses the implementation of the standard distribution SC/CC, thus it will only be briefly explained in Section 3.2.

3.1 Platform Implementation

The CMC platform has been implemented in Java[1], where the SC/SCI structure corresponds nicely to the package structure in the Java programming language. By using the Java package structure, each SC can easily be expanded or replaced without effecting the remaining system. Also, it makes it possible to test each SC in a modular manner in order to measure latency or to verify the logic of the system implementation. In the CMC platform, this becomes necessary according to the increased complexity caused by the interaction among SCs. Currently, the CMC platform is constituted by the following Java packages:

- speech (`dk.lacomoco.cmc.speech`)
- conversation (`dk.lacomoco.cmc.conversation`)
- positioning (`dk.lacomoco.cmc.positioning`)
- distribution (`dk.lacomoco.cmc.distribution`)

Analogous, the distribution CC is represented by the following Java package used by the LBS client:

[1]The current implementation of the positioning TC is based on a Java SDK provided by the Ekahau Positioning Engine.

- distribution (`dk.lacomoco.pda.distribution`)

In order to make the CMC platform generic, dynamic class loading has been used in the implementation. This approach makes it possible to implement new SCs without changing a single line of code in the existing CMC platform. This feature is handled by the configuration file, `cmc.properties`, that among other properties also configures what classes to use by the Java Virtual Machine. In this case, the classes `TCPDistribution`, `CPMLConversation`, `RenderedSpeech` and `EkahauPositioning` have been configured, but this can easily be reconfigured, whenever new SC implementations should replace the old ones (e.g. in this `cmc.properties` file, the `EkahauPositioning` can be replaced by the `DebugPositioning`):

```
distribution_class = dk.lacomoco.cmc.distribution.TCPDistribution
conversation_class = dk.lacomoco.cmc.conversation.CPMLConversation
speech_class = dk.lacomoco.cmc.speech.RenderedSpeech
positioning_class = dk.lacomoco.cmc.positioning.EkahauPositioning
# positioning_class = dk.lacomoco.cmc.positioning.DebugPositioning
```

Thus, the code line below will load the proper implementation of the Speech SC configured by the `cmc.properties` file. If a certain speech implementation does not exist, an exception will be thrown according to the wrong configuration.

```
Speech speech = SpeechCreator.createSpeech(Property.getStringProperty("speech_class"));
```

The `cmc.properties` file is also used to control a lot of other system settings such as database configuration, network timeout and directory configuration (see Appendix **??**).

3.1.1 Speech

The speech SCI declares a collection of QB methods in order to retrieve the synthesized speech. The `getSynthesizedSpeech` method either returns a synthesized speech based on a text input, audio format (*.wav, *.mp3 etc.), sample (kHz) and quantization (bit), or it returns a synthesized speech based on a text input, audio format (*.wav, *.mp3 etc.) and a maximum filesize.

`openConnection(String hostAddress)`: Opens a connection to the applied speech engine using the parameter hostAddress. An exception is thrown, if the connection to the speech engine cannot be established.

`closeConnection()`: Closes the connection to the applied speech engine.

`getSynthesizedSpeech(String text, String format, int sample, int quantization)`: Returns a synthesized speech based on a text input, audio format (*.wav, *.mp3 etc.), samle (kHz) and quantization (bit). An exception is thrown, if an unexpected error occurs in the ongoing processing of synthesized speech.

`getSynthesizedSpeech(String text, String format, int filesize)`: Returns a synthesized speech based on a text input, audio format (*.wav, *.mp3 etc.) and a maximum filesize. An exception is thrown, if an unexpected error occurs in the ongoing processing of synthesized speech.

Currently, the speech SC does not implement a speech synthesizer, as a collection of audio files have been pre-rendered according to the XML documents (cf. `file` attribute of the `agent` tag). Thus, the two different `getSynthesizedSpeech` method declarations all refer to the same implementation, where a collection of pre-rendered audio files simply are mapped as byte arrays. It must be emphasized that this implementation of a speech SC can be replaced due to the Java/C++ SDK provided by the AT&T Natural Voices.

By the same reason, the EB method does not provide any kind of status report in order to notify enrolled observers about the status report concerning the processing of synthesized speech. Still, these methods are declared in the speech SCI according to future developments of the speech SC.

`addStatusListener(StatusListener sl)`: Registers the observer in order to receive status about synthesized speech processing.

`removeStatusListener()`: De-registers the observer.

3.1.2 Conversation

As already explained, the conversation SC only supports static conversations based on XML semi-structured documents. When a conversation is initialized, the XML document is loaded into the Java VM, where a certain *parser* object confirms the file format and breaks the XML document into individual elements. These elements are processed according to the scenario described in Section 2.1.2.

The first two QB methods declared in the conversation SCI, `openConnection(String hostAddress)` and `closeConnection()`, have not been used in this prototype implementation, as no remote conversation engine has been required. Still, the methods are declared in the conversation SCI according to future developments.

The next two QB methods, `getResponse` and `setResponse`, are both essential. As the method names reveal, the first one gets the user response based on a currently loaded XML document, and the second one sets it accordingly. In both cases the XML document is referred to by the `deviceId` parameter, and an exception is thrown, if no XML document is available, or if the `response` parameter cannot be processed.

`openConnection(String hostAddress)`: Opens a connection to the applied conversation engine using the parameter hostAddress. An exception is thrown, if the connection to the conversation engine cannot be established.

`closeConnection()`: Closes the connection to the applied conversation engine.

`getResponse()`: A response is returned to the currently running conversation. An exception is thrown if no conversation is available.

`setResponse(String response)`: A response is set referring to the currently running conversation. An exception is thrown if no conversation is available or if the response does not refer to a currently running conversation.

In order to perform notifications about available conversations, the conversation SCI declares one EB method, where the enrolled observer is notified based on the processing of a specific XML document.

`addConversationListener(ConversationListener cl)`: Registers the observer in order to receive continuously response notification due to an ongoing conversation.

`removeConversationListener()`: De-registers the observer.

The Conversation SC uses a database in order to retrieve XML documents based on user locations, where the filename of each XML document is associated with a certain location as in figure 2.6 on Page 11. This association is utilized by the `conversation` table:

```
create table conversation
( conversationID int unsigned not null auto_increment primary key,
  conversationXML char(30) not null,
  locationID char(30) not null );

insert into conversation values
( null, "4c01.xml", "4c01" ),
( null, "4c02.xml", "4c02" ),
( null, "4c03.xml", "4c03" );
```

Whenever, the Conversation SC receives a notification from the Positioning SC (utilized by the `TrackingListener` interface), a query like the one below will be sent to the database in order to retrieve the right conversational pattern.

```
select conversationXML from conversation where locationID = "4c01"
```

3.1.3 Positioning

The WLAN based Ekahau Positioning Engine 3.1 runs on a central server at the IT University of Copenhagen, and a socket connection can be established internally by using the enclosed SDK. This connection is handled by the `openConnection` method. In addition to that, the positioning SCI declares four QB methods in order to retrieve information about tracked devices and locations. The `getTrackedDevices` method returns a collection of tracked devices according to the specific `area`. In order to receive all tracked devices connected to the positioning engine, the method is called with no argument. The `getLocation` method returns a location (reffering to the `deviceId`) contained in a 4-tuple, `pos(x, y, z, area)`, representing a physical location as well as a logical location (cf. `area`). Also, a collection of locations can be retrieved due to a collection of deviceId's through the `getLocations` method.

`openConnection(String hostAddress)`: Opens a connection to the applied positioning engine using the parameter hostAddress. An exception is thrown, if the connection to the speech engine cannot be established.

`closeConnection()`: Closes the connection to the applied positioning engine using the parameter hostAddress.

`getTrackedDevices()`: Returns a collection of all tracked devices connected to the applied positioning engine.

`getTrackedDevices(String area)`: Returns a collection of tracked connected to the applied positioning engine according to the specific area.

`getLocation(String deviceId)`: Returns a location (reffering to the deviceId) contained in a 4-tuple[2] `pos(x, y, z, area)`, where x, y and z are representing the physical location and area represents the logical location.

`getLocations(String[] deviceId)`: Returns a map of locations (reffering to the deviceIds) contained in a 4-tuple `pos(x, y, z, area)`, where x, y and z are representing the physical location and area represents the logical location.

The declared EB method, `addTrackingListener`, ensures that the the registered observers are notified about location data updates from the positioning engine.

`addTrackingListener(String deviceId, TrackingListener tl)`: Registers the observer (cf. deviceId) in order to receive continuously notification about recorded locations.

`removeTrackingListener(String deviceId)`: De-registers the observer (cf. deviceId).

3.1.4 Distribution

The distribution CC implements a collection of QB methods in order to establish a connection to the distribution SC at the CMC platform and to establish the application protocol. The `openConnection` method opens a socket connection to the distribution SC running on a server with a specified `hostAddress` (configured in the `pda.properties` file), if the internal connection to the positioning, conversation and speech SC can be established; otherwise, the distribution SC cannot be initialized, and an exception is thrown.

The `transmitAuthentication` method transmits the authentication to the distribution SC that will query a database in order to authenticate the combination of username, password and deviceId. The query result is then transmitted back to the distribution CC. Finally, the `transmitResponse` method transmits the user responses based on the LBS client input.

`openConnection()`: Opens a connection to the applied CMC platform. An exception is thrown, if the external connection to the conversation SC cannot be established.

`closeConnection()`: Closes the connection to the applied CMC platform.

`transmitAuthentication(String username, String password, String bluetoothID, String phoneID)`: Authenticates with the CMC platform.

`transmitResponse(String response)`: Transmits the response from the distribution CC to the distribution SC on the CMC platform.

In order to receive notifications about authentication results and incomming data packages, the EB methods, `addAuthenticationListener` and `addDistributionListener`, have been implemented as well.

`addAuthenticationListener(AuthenticationListener al)`: Registers the observer in order to receive the authentication result.

`removeAuthenticationListener()`: De-registers the observer.

[2]A tuple represents a data object containing other objects (of different types) as elements.

`addDistributionListener(DistributionListener dl)`: Registers the observer in order to receive notification about incomming data packages.

`removeDistributionListener()`: De-registers the observer.

The implementation behind the distribution CC methods has been hidden due to its complexity. It covers the logistic of the application protocol used by the distributed system introduced by the CMC platform. It is outside the scope of this report to go through the entire application protocol. Instead, the reader is referred to the code in the two distribution packages (cf. Section 3.1) representing the client-/server-side.

In order to provide authentication on the server-side, the system uses a simple databases structure. In this case the `administrator` table has been created:

```
create table administrator
( userID int unsigned not null auto_increment primary key,
  userName char(30) not null,
  password char(30) not null );

insert into administrator values
( null, "Administrator", "cmc" );
```

Furthermore, a user can only log on, if he/she has been inserted into the `user` table due to the user authentication. When the user logs on, a specific user profile is updated according to a certain address (IP, Bluetooth etc.) on the handheld device. These addresses are used by the positioning engine in order to identify a connected handheld device. Notice, only by manually inserting values into the `user` table, other users will be able to log on, though the creation of new user profiles should be made by the users themselves through a web interface. However, this has been considered outside the scope of the development of a first CMC platform prototype.

```
create table user
( userID int unsigned not null auto_increment primary key,
  username char(30) not null,
  password char(30) not null,
  ip char(30),
  bluetoothID char(30) );

insert into user values
( null, "Administrator", "cmc", "", "" );
```

In Section 2.1.4 it was suggested to reduce the transmission of byte arrays by caching already transmitted byte arrays on the client-side. This look-up service has not been implemented, thus it can be seen as a future development. Instead, the application protocol provides a switching functionality that decides whether the byte array has to be sent or not. If all audio files have been transmitted beforehand to the handheld device, the server-side only has to send a much smaller data package (representing the filename) in order to trigger a specific audio file on the client-side. This functionality can be (and has been) used, whenever the system is tested in a WLAN with limited and varying bandwidth. The switching functionality can be controlled by configuring the `cmc.properties` file, where *true* will turn on the byte array distribution.

As it was suggested in Section 2.1.4, the LBS client will appear more user-friendly, if the user is well-informed about the network status. A simple solution to this problem has been to implement a timer thread that starts, whenever the streaming of an audio file is expected on the client-side. If a certain latency is exceeded, the timer thread will notify the registered observer (cf. `DistributionListener`) in order to take the right precautions such as playing a specific audio file excusing the latency.

3.2 LBS Implementation

In order to meet the low power consumption on the handheld device, the current LBS client implementation mainly provides a simple GUI showing a collection of available user responses according to the XML document. Also, the LBS client uses the distribution CC in order to utilize the standard distributed system of the CMC platform.

As the Connected Device Configuration (CDC) defined in J2ME is not supported by the handheld devices[3] used by this system, it has been a problem to play simple files in a Java program. The JVM on these handheld devices is a very limited version of the normal Java SDK (cf. JDK 1.1.8) that does not support the playing of audio files [12]. Therefore, it has been necessary to use the Java Native Interface (JNI) in order to built-in this functionality provided by a dll file. In order to create this dll file, a C++ program has been implemented by using *Microsoft eMbedded Visual C++*. The actual code (cf. `AudioMonitorWinCE.h` and `AudioMonitorWinCE.cpp`) can be found in the `CMC/client/src/dll` folder.

In order to utilize this functionality (play and stop playing), the Java program simply loads the `AudioMonitorWinCE.dll` and uses the two native methods `playWinCE` and `stop-WinCE`.

[3]Currently, the system has been tested on a Pocket LOOX 610 and Compaq iPAQ h3950.

Chapter 4

Platform Demonstration

The CMC platform prototype has been tested at the IT University of Copenhagen, where the Ekahau Positioning Engine 3.1 is currently running. Furthermore, the distribution CC has been imported to the handheld device in order to test the standard distributed system of the CMC platform. Thus, the LBS client (used in this demonstration) is trivial, as it does not contribute to the LBS functionality but merely utilizes the standard functionality of the distribution CC.

The LBS client establishes a connection to the CMC platform by initiating the `Distribution` object (see distribution CC), and before a real session can be started, the LBS client will have to authenticate through the authentication GUI using the `GhostAuthentication` object. The necessary server information (such as IP and port number) has been configured in the `pda.properties` file.

Provided that the functionality of the different SCs can be retrieved, the session will start, when the user has been sucessfully authenticated. Depending on the location of the handheld device, a specific conversational pattern (cf. XML document) will now be initiated in order to start the conversation between the user and system. In this demonstration, the following two conversational patterns will be shown:

1. The static conversational pattern shows, how a static conversation can be assigned to a certain room (or area) in the building and how the user response is used in order to process the XML document.

2. The dynamic conversational pattern uses the continuously update of location data in order to process the XML document.

A small collection of both static and dynamic XML documents have been assigned to certain areas at the IT University of Copenhagen, thus it has been possible to test the system in different scenarios. Each one of these scenarios are examples on how different LBSs can assist users in different situations depending on specific locations. These scenarios correspond to the main ambitions in the DELCA[1] project, where the intention is to combine context specific conversation mediated assistance with an interactive narrative universe designed to promote both pleasant and efficient service[2]. In the DELCA project, the user will be able to communicate directly to a collection of conversational agents (or ghosts), who will respond with a meaningful

[1]The basic concepts of the DELCA (dis-embodied location-specific conversational agents) project was introduced in Chapter 1 on Page 2.

[2]Articulated by some of the associated people in the working paper ”Spooky Virtual Agents: Enter the World of Ghosts” [9].

answer. These ghosts merely correspond with the processing of conversational patterns discussed throughout this report, thus the DELCA project nicely illustrates the main functionality of the CMC platform.

4.1 DELCA Demonstration

In the following demonstration, the system has been tested on one handheld device. The user has been authenticated through a simple authentication GUI, and the session has been started. Whenever the user goes to a specific area that contains an XML document, the conversation will start. Through the loudspeakers the user will now hear the ghost response as a synthesized speech, and the user can respond by pressing one of the available buttons in a simple GUI.

In one of the DELCA scenarios the user enters a printer area, where he/she is confronted with a ghost, who eagerly tells all about xerox and printer facilities. The XML document below demonstrates, how this scenario has been implemented:

```
<conversation id="event">
  <ghost file="printer_1_1.wav" text="Hallo Thomas. Do you need assistance?">
    <user respond="Yes">
      <ghost file="printer_1_2_yes.wav" text="Are you going to xerox or print?">
        <user respond="Xerox">
          <ghost file="printer_1_3_xerox_1.wav" text="Ghost: Now put the original in the Xerox tray. Can you find it?"/>
        </user>
        <user respond="Print">
          <ghost file="butler_1_3_print_2.wav" text="..."/>
        </user>
      </ghost>
    </user>
    <user respond="No">
      <ghost file="printer_1_2_no.wav" text="Okay. Good luck."/>
    </user>
  </ghost>
</conversation>
```

When the positioning engine registers a user entering the printer area, the scenario starts to play the synthesized speech referred in the first element node in the XML document: ”Hallo Thomas. Do you need assistance?” The system now awaits the user response and then continues to process the XML document. Depending on the user response, a scenario could be the following:

```
Ghost: Hallo Thomas. Do you need assistance?
User: Yes
Ghost: Are you going to xerox or print?
User: Xerox
Ghost: Now put the original in the Xerox tray. Can you find it?
```

If the user leaves the printer area during a conversation, the conversation will terminate immediately, as the conversation only is relevant due to that specific area. The XML document can be extended in order to provide much longer conversations; in this case a short one was used in order to illustrate the idea.

Beside the static conversational document, also the dynamic concept of the conversational pattern has been tested by guiding the user from a specific location to a pre-defined destination. Although some fluctuating latency has been experienced (caused by the positioning engine update), it is possible to perform a simple guiding feature based on a dynamic conversational pattern.

```
<conversation scenario="dynamic">
  <agent file="4Cc1.wav" text="Please, walk down the long corridor.">
    <user response="4Cc2">
      <agent file="4Cc2.wav" text="Please, continue straight out.">
        <user response="4Cc3">
          <agent file="4Cc3.wav" text="Please, do continue down the long corridor.">
            <user response="4Cc4">
              <agent file="4Cc4.wav" text="The door leading to room 4C03 is on your right hand."/>
            </user>
          </agent>
        </user>
        <user response="4C01">
          <agent file="4C01.wav" text="You where supposed to continue down the long corridor."/>
        </user>
      </agent>
    </user>
  </agent>
</conversation>
```

In this case, the scenario will depend on the user location, as the attribute `scenario` states (cf. dynamic conversational pattern). The current XML document will only start to process, when the user enters the `4Cc1` area. Also, the scenario will depend on a very predictable user behaviour like the following:

```
Ghost: Please, walk down the long corridor.
User: 4Cc2
Ghost: Please, continue straight out.
User: 4Cc3
Ghost: Please, do continue down the long corridor.
User: 4Cc4
Ghost: The door leading to room 4C03 is on your right hand.
```

Obviously, the dynamic approach is very limited, as the XML document only represents a determined set of interrelated locations. Therefore, the guiding feature will apply to some pre-defined routes, thus the user can only use the service starting from very specific locations. Furthermore, the system cannot predict the many possible directions, the user might choose to go, and it would certainly be a huge task to include them all in one XML document.

4.2 DELCA Scenarios

As already mentioned in Section 2.2, the DELCA project presents a comprehensive collection of conceptual ideas within the area of ubiquitous computing. Therefore, the demonstration above only shows the general concepts of the DELCA project provided by the CMC platform prototype. Merely, the DELCA project can be described as a multi location-based service (MLBS), where each service represents a specific context-dependent scenario performed by ghost characters; mainly characterized by their voices (cf. synthesized speech). The printer scenario (that was shown above) were showing one of these ghost charaters called the *printer ghost*. Other ghosts have been suggested, but the very general idea is to develop LBSs in a more personalized way utilizing conversational patterns as shown by the CMC platform.

Chapter 5

Conclusion and Perspectives

In Chapter 1 a collection of requirement specifications where put in order to see how well a context-dependent mobile communication platform can be made based on existing technology. Still, a lot of work needs to be done, if such a platform may support the many different requirements of different kinds of LBSs.

5.1 Conclusion

The collection of SCs in the CMC platform prototype have been designed and implemented in order to support some basic features of LBSs. The design and implementation of the CMC platform facilitates development of LBSs that base themselves mainly on speech interaction. Based on the definition of an AUI (cf. Chapter 1 on Page 2), the CMC platform is only half-way, as it only integrates speech synthesis and leaves speech recognition as a future development. Thus, the system will neither be fully hands-free nor eyes-free until a speech recognizer has been developed for medium devices such as handheld devices.

The CMC platform is designed in a modular way, allowing new technology components to be added easily, and a prototype LBS has been implemented to demonstrate the main features. Also, field trials have shown that the CMC platform is able to support LBSs in a very general way.

5.2 Perspectives

Based on the system requirements and the extension of functionality, a lot of perspectives towards the CMC platform can be formulated. A few of these perspectives are presented below.

Obviously, LBSs are dependent on stable location data, though the positioning SC might be extended in order to include filtering algorithms to ensure a higher probability of correct location data as well as prediction algorithms based on historical data. The filtering algorithm may remove location ’noise’ such as impossible jumps between floors and rooms etc. Prediction can be important, if e.g. the notification on location data is interrupted. Based on some historical data (representing the normal location pattern), it can therefore be predicted in which direction the user might go. Moreover, an LBS is supposed to work both inside and outside, thus the positioning SC should apply many different positioning technologies simultanously.

Currently, the conversation SC only represents static conversations, as a dynamic conversation will require some artificial intelligent chat-robots. A lot of computations must go into providing such a dynamic conversation, as the system must be able to compute a meaningful reply based on the user input. In other words, the system must be able to analyze the input sentence provided by the user in order to compute the output sentence. The user might ask: ”Where is the nearest coffee maker?” A meaningful answer to that would be: ”Go down the hallway and turn left, when you see the kitchen sign.”. In this example, the system needs to break down the sentence into a collection of computable commands. Subsequently, the result of these computations must be built into an understandable sentence. One computation will be to find a collection of all coffee makers, used by another computation to find the nearest one. In order to provide a guiding feature, the dynamic conversational patterns must be generated based on a shortest path algorithm [4]. On the contrary, the static conversational patterns are pre-defined, thus less flexible.

Another aspect of usability is on how to guide users inside a building. Conventionally, guiding has been performed by saying ”turn left now”, ”continue straight on” etc., but in practise this causes problems when combined with positioning inaccuracy, as the user might take a turn too soon or too late. Instead, *landmarks* should be utilized to inform the user of locations of path changes; they are used in expressions like ”turn left after the coffee vending machine” and ”please continue through the door at the end of the hall”.

Appendix A

Targeted Hardware

In order to present a system running in a WLAN, the targeted hardware will be PDAs on the client-side as well as a standard PC and laptop on the server-side. Currently, the client-side has been tested on a Fujitsu Siemens Pocket LOOX 610 (400.0 MHz processor (Intel XScale XPXA250) speed, 64 MB installed ROM, 64 MB installed RAM, Bluetooth, IEEE 802.11b WiFi, 240 x 320 pixels touchscreen, built-in speaker and Windows Mobile 2003) and a Compaq iPAQ h3950 (400.0 MHz processor (Intel XScale XPXA255) speed, 32 MB installed ROM, 64 MB installed RAM, 240 x 320 pixels touchscreen, built-in speaker and Windows Mobile 2002) using an external 11 Mbit/s data rate PCMCIA card (AGERE ORINOCO Gold/Silver Classic (b)/AGERE Avaya Wireless World Card Gold/Silver (b)). The server-side has been tested on a DELL LATITUDE D800 (1.3 GHz processor speed, 128 MB RAM, 20 GB hard drive capacity, 32 MB video memory and Microsoft Windows XP Professional) laptop, and the Ekahau Positioning Engine 3.0 runs on a standard server PC at the IT University of Copenhagen, where a calibration model covering a selected test area has been configured. The engine has an update frequency every other second and a maximum limit on 100 connected devices.

Bibliography

[1] *AT&T Natural Voices Text-To-Speech Engines. System Developer's Guide*, 2004.

[2] Reza B'Far. *Mobile computing principles. Designing and developing mobile applications with UML and XML*. Cambridge University Press, 2005.

[3] BlipSystems. *BlipNet 3.0. Application Developers Tutorial*, 2004.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[5] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems. Concepts and Design*. Addison-Wesley, 2001.

[6] Ekahau. *Ekahau Positioning Engine 3.0. Developer Guide*, 2004.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] Ulf Leonhardt. Supporting location-awareness in open distributed systems. Master's thesis, University of London, 1998.

[9] Mikkel Holm Sørensen and John Paulin Hansen. Spooky virtual agents: Enter the world of ghosts. 2004.

[10] John Aa. Sørensen, Kåre J. Kristoffersen, Anders Cervera, Martin Schiøtz, Thomas Lynge, Zoltan Safar, and Lars Birkedal. An infrastructure for context dependent mobile multimedia communication. In *The 2004 IEEE Workshop on Multimedia Signal Processing Sep. 29 - Oct. 1*, pages 462–465, 2004. ISBN: 0-7803-8579-9.

[11] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM archive, Volume 36, Issue 7*, July 1993.

[12] Daryl Wilding-McBride. *Java. Development on PDAs. Building Applications for Pocket PC and Palm Devices*. 2003.