Context-Aware Services for Constrained Mobile Devices

Karel-Henk Nijhuis

University of Twente

Faculty of Electrical Engineering,
Computer Science and Mathematics

Architecture and Services of
Network Applications Chair

Date: July 25th, 2007

Graduation Committee:
Dr. L. Ferreira Pires
P. Dockhorn Costa MSc
Dr. Marten van Sinderen
Preface

I would like to thank Luís, Patricia and Marten for their neutral and supportive comments on my thesis and research project. I really admire the way they have stimulated me and showed me how to improve my accurateness. During the graduation project I learned that keeping a clear vision of the research goals is essential to its success, because this process can easily divert. I also experienced that scientific writing is an important skill that requires preciseness and brevity, which I had to internalize during this project.

I want to thank my parents, family and friends for keeping confidence in me, at the times that this project has been a struggle. Rounding up this project and my Computer Science study at the University of Twente provides me with a new and solid foundation for my future.
Abstract

With the emergence of new browsing technologies for mobile phones and mobile network providers shifting from pay-per-megabyte to flat-rate pricing for data transfer (Jaokar and Fish 2006), data services for mobile phones start to become more popular. New applications that use distributed services will emerge and allow users to retrieve any kind of information like the latest news headlines, stock quotes and digital media. However, finding and navigating the information from a mobile phone, that typically has a small screen and an inconvenient keypad can be cumbersome.

The context-awareness paradigm helps to discover information that might be interesting for a user based on the situation the user is in. In this thesis we explore how the development of applications for constrained mobile devices could benefit from a proactive context management framework that uses service-orientation.

To illustrate the capabilities of this framework we developed an application scenario of a context-aware calendar application that uses a concert recommendation service implemented using context sources. We evaluate the role of the context management framework and show that our approach can help to create portable and resource efficient applications.
Acknowledgements

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025.
# Table of Contents

1 Introduction ....................................................................................................................... 7  
1.1 Motivation ..................................................................................................................... 7  
1.2 Objectives .................................................................................................................... 8  
1.3 Approach ...................................................................................................................... 8  
1.4 Structure ..................................................................................................................... 9  

2 Context Modeling ............................................................................................................. 11  
2.1 Context-awareness ....................................................................................................... 11  
2.2 Main characteristics of context-information .................................................................. 12  
2.3 Conceptual foundations for the AWARENESS architecture ....................................... 13  
2.4 The Event-Control-Action architectural pattern ......................................................... 16  

3 The AWARENESS CMF Architecture ............................................................................ 18  
3.1 Requirements ............................................................................................................. 18  
3.2 Components .............................................................................................................. 19  
3.3 Context sources ........................................................................................................... 19  
3.4 The AWARENESS Ontology ..................................................................................... 22  

4 The context-aware recommendation service development ........................................... 24  
4.1 Application scenario .................................................................................................... 24  
4.2 Development process ................................................................................................. 25  
4.3 Service requirements ................................................................................................. 26  
4.4 Design of the context sources .................................................................................... 27  
4.4.1 User profile data model ......................................................................................... 27  
4.4.2 The announcement context source ...................................................................... 29  
4.4.3 The location context source ................................................................................ 30  
4.4.4 The availability context source ........................................................................... 31  
4.5 Service description .................................................................................................... 32  
4.5.1 Availability-based recommendation .................................................................... 33  
4.5.2 Location-based recommendation ....................................................................... 33  

5 The context-aware calendar development ...................................................................... 35  
5.1 Requirements ............................................................................................................ 35  
5.2 The Java ME application platform ............................................................................. 36  
5.2.1 General characteristics ......................................................................................... 36  
5.2.2 The Mobile Information Device Profile ............................................................... 36  
5.2.3 Basic CLDC/MIDP distributed programming model ........................................... 37  
5.2.4 MIDP application types ....................................................................................... 38  
5.3 Design ....................................................................................................................... 39  
5.3.1 The application model .......................................................................................... 39  
5.3.2 The local data model ............................................................................................ 40  
5.3.3 The notification handler ...................................................................................... 41  

6 Implementation details ................................................................................................. 43  
6.1 The concert announcement context source .............................................................. 43  
6.2 Generating Jena interfaces using Jastor ..................................................................... 45  
6.3 Communication with the CMF ................................................................................. 46  
6.4 Application-specific client implementation ............................................................... 47  

7 Conclusions ................................................................................................................... 48
7.1 Distributed context management................................................................. 48
7.2 Context information might improve mobile browsing.......................... 48
7.3 Shift in mobile application types................................................................. 49
7.4 Future work ............................................................................................... 49
Appendices ....................................................................................................... 52
Appendix A – Use cases of the recommendation service.............................. 53
Appendix B – Use cases of the context-aware calendar................................. 59
Appendix C – AnnouncementCS.owl.............................................................. 64
Appendix D – Jastor build task for generating OWL interfaces..................... 66
1 Introduction

This chapter presents the motivation, objectives and structure of this thesis. This chapter is further structured as follows: Section 1.1 briefly presents the motivation of this work, Section 1.2 states the objectives of this thesis, Section 1.3 presents the approach adopted in the development of this thesis and Section 1.4 outlines the structure of this thesis by presenting an overview of the chapters.

1.1 Motivation

Mobile phones and Personal Digital Assistants (PDAs) are converging into mobile lifestyle devices that offer a wide range of mobile applications to end-users (van Sinderen, van Halteren et al. 2006). These applications are no longer limited to the boundaries of the mobile device, but they will increasingly exploit the resources available through ubiquitous networks to realize seamless personalization of these devices. An important requirement for the success of these applications is their ability to adapt themselves to the situation the user is in, commonly referred to as context-awareness.

Context-awareness refers to the capability of a software application to provide services to their users based on the user’s current context. It has emerged as an important and desirable feature in distributed mobile systems, since it benefits from the changes in the user’s context to dynamically tailor services to the user’s current situation and needs (Dockhorn Costa, Šuizzardi et al. 2006). Whether a user is at home, at work or with friends, context-awareness will help to tailor services according to the situation the user is.

The Context Management Framework (CMF) developed within the Freeband AWARENESS project (van Sinderen, van Halteren et al. 2006) is a framework that aids the development of context-aware services and applications. It includes modules that facilitate distributed access and sharing of so-called context sources that ‘sense’ the environment of a user.

The Freeband program addresses the communications knowledge chain in the direction of the new ubiquitous communication paradigm. Based on this vision, the key research questions are related to three main themes (Freeband AWARENESS 2005):

- Society, Users and Applications: what are the new possibilities in different sectors for ubiquitous communication and ambient intelligence and how can they be realized?
- Networking, Service Provisioning and Generic User Interaction: the telecommunication infrastructure viewed from the user's perspective. How can services be offered to end-users such that they be accessed in a convenient and user-friendly way?
- Enabling Technologies: no new services emerge without adequate technology; conversely, it is the technology that drives the new paradigms. Which technology is required to enable these services?
We are particularly interested in how context-aware applications for constrained mobile devices could be realized using a service-oriented context management framework. A service-oriented approach to context-awareness for mobile devices might enable a new class of mobile applications that can pro-actively provide information that a user is interested in.

1.2 Objectives

Our objective is to evaluate the service-oriented approach and the use of a context management framework for constrained devices. Service-orientation could help to overcome limitations of these devices by, e.g., distributing program execution and might in this way create a new target platform for context-aware applications.

This has led to our research question:

| How can service-oriented architecture contribute to the development of resource efficient context-aware applications for constrained mobile devices? |

Context management frameworks like the CMF offer an environment that is optimized for the deployment of services that act and reason with context information. Since it integrates the service-oriented approach with facilities for context management we will take this framework as the starting point of our evaluation.

1.3 Approach

In order to find an answer to this research question we took the following approach:

- Investigate context modeling concepts, context-awareness architecture and applications and learn about their purpose and properties that specifically support the development of context-aware applications,
- Study the CMF specification and its current prototype,
- Study the capabilities and extensions of the Java platform for CLDC/MIDP devices,
- Define an application scenario that exhibits the properties that were determined for context-aware mobile applications,
- Develop a prototype and define the development approach for implementing the requirements derived from the application scenario,
- Evaluate the CMF with respect to the research questions.
Figure 1 relates these steps and distinct steps in which background information is studied and the steps that constitute our own activities in order to find an answer to the research question that we posed.

![Figure 1 Overview of our approach](image)

1.4 Structure

The structure of this thesis reflects the issues that have been dealt with throughout our research process. Figure 2 gives an overview of the contents of the chapters containing background information on the basic principles of context-awareness and context modeling (Chapter 2), and the functional structure of the CMF architecture (Chapter 3).

![Figure 2 Chapters containing background information on context modeling and the CMF](image)
Figure 3 shows the structure of the chapters that deal with the design and implementation of the proposed application scenario.

Chapter 4 introduces an context-aware application scenario and discusses the design of the CMF components that we have defined for this application. Chapter 5 discusses the CLDC/MIDP application platform and the design of the context-aware calendar application that depends on the CMF and the recommendation service. Chapter 6 discusses the implementation details of the major topics that we have dealt with while working on the context-aware calendar prototype. Finally, chapter 7 presents our conclusions and recommendations for future work.
2 Context Modeling

In this chapter we introduce the conceptual foundations of context modeling that have been applied within the Freeband AWARENESS project. Section 2.1 introduces some of basic context types that context-aware systems deal with. Section 2.2 introduces the main characteristics of context information. Section 2.3 contains a discussion on the development of the conceptual foundations that influenced the Freeband AWARENESS architecture. In section 2.4 we discuss the Event-Control-Action pattern (Dockhorn Costa, Ferreira Pires et al. 2005) and how situation modeling could help specify the behavior of context-aware systems.

2.1 Context-awareness

Context-awareness refers to the idea that distributed applications can both sense, and react to changes in their environment. These applications may have information about the circumstances under which they are able to operate, and based on rules, react accordingly (Schilit, Adams et al. 1994). Context-awareness is often applied in ubiquitous or pervasive computing, which is a computing model in which computer functions are integrated into everyday life, often in an invisible way.

The model of the environment that context-aware applications work with is called a context model. Context models can hold all kinds of information and may be categorized into context related to human factors and context related to the physical environment of a user.

Context related to human factors can be structured into three categories (Schmidt, Beigl et al. 1999):

- information about the user (knowledge of habits, emotional state, biophysiological conditions, etc.),
- the user’s social environment (co-location of other people, social interaction, group dynamics, etc., and the user’s tasks (spontaneous activity, engaged tasks, general goals, etc.).
- the user’s tasks (goal-directed activities or the general goals of users)

Context related to physical environment can also be structured into three categories:

- location (absolute position, relative position, co-location, etc.),
- infrastructure (surrounding resources for computation, communication, task performance), and
- physical conditions (noise, light, pressure, etc.).
2.2 Main characteristics of context-information

When dealing with context information the following characteristics have to be kept in mind:

- Context-information comes from heterogeneous sources
- Context-information might need to be enriched to be meaningful
- Context information can be contradictory
- Context-information may applicable only in a certain situation
- Context-information is continuously subject to change

The example below illustrates these characteristics and shows how context-awareness could enrich a car navigation system by integrating context information that comes from the environment of the user.

Location-Based Services (LBS) form a class of context-aware services that uses location context to determine the behavior of an application. A car navigation system is an example of such a system; it guides a driver from his current location to a requested destination using location information that is determined using a GPS receiver. Using a set of electronic maps driving directions are provided to the driver.

In a context-aware system context information is typically combined with other types of (context) information to infer higher level knowledge. For example, a car navigation system could be extended with a function to find cheapest gas station in the area. To this end it could access a database with gas stations and their fuel prices and a fuel level sensor could indicate when to consult this database.

In context-aware systems context-information might come from heterogeneous context sources: The car navigation system combines information coming from its internal GPS receiver, the car’s fuel sensor and an online database containing fuel prices.

If the fuel sensor indicates that the fuel tank is empty, but the car is still driving, we deal with contradictory sensory information. It is possible that the fuel sensors do not indicate the right fuel level, which results in the navigation system to keep notifying about gas stations in the area. These notifications are not interesting if the driver has just passed by gas station. Therefore, context information may be inconsistent or incomplete. In case a GPS sensor fails, location information is not updated and the car navigation system has to reason with outdated information.

Context information may only be applicable in a certain situation. If the car is parked in front of the driver’s house after a long day of work, the driver might not be ready to receive any messages that indicate that he has to go to a gas station.

Another characteristic of context information is that it is continuously subject to change. The location of the car and the fuel level changes while driving. Since the user lives in a dynamic environment it can be expected that context changes while a user is on the move. However, the frequency at which context information changes is often determined by the type of context and the availability of context-sensors.
2.3 **Conceptual foundations for the AWARENESS architecture**

Context-aware services deal with context information coming from heterogeneous and typically distributed sources. To be able to deal with this information in an efficient way, the information that is exchanged within a sensor network should have consistent semantics. The semantics of a formal vocabulary can be expressed using ontologies.

According to (Guarino 1998) an ontology is the logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world.

![Figure 4 Ontology approximating the set of intended models (Guarino 1998)](image)

The intended models of a logical language using such a formal vocabulary are constrained by its ontological commitment. Ontologies indirectly reflect this commitment (and the underlying conceptualization) by approximating these intended models (Figure 4).

In (Guarino 1998) it is stated that ontologies should be considered in the light of both the methodological and architectural approach. On the methodological side, the main peculiarity is the adoption of a highly interdisciplinary approach, where philosophy and linguistics play a fundamental role in analyzing the structure of a given reality at a high level of generality and in forming a clear and rigorous vocabulary. On the architectural side, the most interesting aspect is the centrality of the role that an ontology can play in an information system, leading to ontology-driven information systems.

Figure 5 illustrates one of the main challenges in information integration. Two systems A and B, that use language L can only communicate if their set of intended models I(L) overlap. Determining this overlap in two systems that need to be integrated, becomes is facilitated in case both systems share an ontology.
(Dockhorn Costa, A. Almeida et al. 2005) discusses the methodological side of a context-aware ontology relating to the development of foundational ontologies, which are in line with conceptual theories in the areas of philosophy and cognitive sciences.

The main concepts shared within context-awareness modeling are the concepts of *entity* and *context*. Context can be defined as the interrelated conditions in which something exists. (Dockhorn Costa, A. Almeida et al. 2005), which subsumes the existence of an entity. An entity is the bearer of context.

In Figure 3 shows the relationship between entity and context is depicted. It also shows a possible refinement of context and entity that has been proposed by (Dockhorn Costa, A. Almeida et al. 2005).

Figure 6 shows a refinement of the entity class. We find an entity denoted as a physical entity (i.e., an object that exists in the real world). Figure 3 shows a device and a person as possible examples of a physical entity. It also shows a so-called container entity that is
capable of containing other entities. This concept could be used, e.g., to model a person that is contained in a room, which could be a situation of interest to context-aware applications.

In (Dockhorn Costa, A. Almeida et al. 2005) we also find a distinction between intrinsic and relational context. Whereas intrinsic context is inherent to a single entity like the temperature of an entity, relational context inheres within the relationship between two or more entities. The friendship between two people is considered as an example of relational context.

Figure 7 shows a context model of the car navigation example. Using UML stereotypes we have indicated to which metaclass (entity or context) the car navigation objects relate.

![Diagram](image)

**Figure 7 A context model for the car navigation system**

Situations can be defined by creating instances of this model. A situation is defined as a particular state of affairs that is of interest to an application (Dockhorn Costa, Guizzardi et al. 2006). A situation can be viewed as an instance of a context model, which holds snapshot of the values of instance variables.
If we consider a car at time $t$ that is driving at the speed of $q_s(t) = 60\text{mph}$, with its fuel tank nearly empty $q_v(t) = 0.1\text{gal}$, and it the car driving at $q_l(t) = 52^\circ13'\text{N}$ (latitude) and $6^\circ54'\text{E}$ (longitude). Using the notation, we can come up with a model for this situation (Figure 8).

### 2.4 The Event-Control-Action architectural pattern

Situations represent states that application act upon. The Event-Control-Action (ECA) architectural pattern provides a high level structure for systems that pro-actively react upon context or situation changes. It has been devised to decouple context managements issues, such as sensing and processing context, from reaction concerns regarding reacting upon context changes, under the control of an application model (Dockhorn Costa, Ferreira Pires et al. 2005).
The architectural pattern is named after its three constituting components. Figure 9 gives an overview of the ECA pattern. A number of context processors bring together context information into a predefined context model. As a consequence of context processing, context processors generate events that may notify the controller. The controller observes the context processors and acts according to the application behavior descriptions. The behavior description consists of a set of condition rules, which associate actions with a particular situation. Finally, if any of the condition rules are satisfied, the controller delegates the execution of the actions prescribed in the selected rules to the action performer.
3 The AWARENESS CMF Architecture

Context management deals with the structured retrieval of context information from distributed heterogeneous context sources. In this chapter we discuss the Context Management Framework (CMF) (Kranenburg, Salden et al. 2005) that provides intelligent facilities for enabling context-aware services. The CMF implements a part of the AWARENESS architecture (van Sinderen, van Halteren et al. 2006). Section 3.1 lists the requirements that have been set for the CMF. In section 3.2 the main CMF components are discussed with respect to their functional responsibilities. In section 3.3 we discuss context sources and the role they play inside the CMF architecture. Section 3.4 motivates the adoption of an ontology for the exchange of context-information.

3.1 Requirements

Context-aware service enablers can assist services and applications to adapt to changing environments, such as available network resources, location updates, and the presence status of nearby buddies. The requirements (Kranenburg, Salden et al. 2005) led to the design and realization of the CMF can be categorized in four categories that relate to the characteristics of context information mentioned in section 2.2.

- **Auto-discovery of distributed context information**
  - Find (and exchange) sources of information, regarding persons, devices, location and presence information (support for a large number of context sources including alternative sources of the same context),
  - Get the information from these context sources (context gathering: transparent use of distributed sources of context and support distributed usage of context information).

- **Support for context reasoning**
  - Context reasoning allows application developers to interpret the context information and derive plausible and usable information needed for the given purpose.

- **Application service provisioning**
  - Follow an approach of separation of concerns between sources of context information and users of context information.

- **Auto-configuration of devices**
  - Support for device auto-configuration: devices with appropriate authorization should be able to subscribe to the context-aware framework in order to gain awareness on other people and objects in the environment.
3.2 Components

The CMF architecture is a flexible component-based architecture that consists of a container that manages the deployment of components. The CMF is build around the concept of context sources (see section 3.3) that provide a uniform interface to components or applications that use the contextual information, hiding the details of the underlying context-sensing mechanisms (Freeband AWARENESS 2005).

Figure 10 shows the CMF container and its main components. The authentication and authorization service is involved with enforcing access control and privacy policies on the components and applications that access context sources that have been registered inside the context sources registry.

The context sources registry contains references to context sources that are either deployed in the same container or inside another CMF container that has exported context sources through RPC. The context sources registry is maintained by a context provider that is used to discover context sources.

![Figure 10 The main CMF components](image)

The service registry stores references to remote services and is able to proxy them if those services allow that. Registered services can be used by CMF service in convenient and transparent way.

The user- and profile manager provides means to access and maintain user- and profile-specific data. User management relates to managing static data, e.g. user name and address, whereas profile management refers to managing dynamic features, like user preferences and settings.

3.3 Context sources

We already mentioned context sources as an important concept that has been adopted by the CMF. Context sources implement the principle of context processors found in the
ECA pattern (Figure 9) and are responsible for retrieving context information and notifying subscribers of context changes. Context sources sense context information from the environment through context sensors.

In section 2.1 we showed that this context information comprises a broad spectrum of information types that can be categorized into human factors related context and physical environment related context. Context sources wrap and publish this context information.

The CMF assumes that a context source yields context information objects, either by directly monitoring context, which is referred to as context wrapping, or by intelligent context-based reasoning with heterogeneous and distributed context information objects (Kranenburg, Salden et al. 2005).

Context-based reasoners may perform additional processing of context information, like storage, merging or filtering. Five types of context-based reasoners are identified by (Kranenburg and Eertink 2005):

- A context storage context source allows for storage, synchronization and retrieval of (inferred or user-specified) context-information objects. Part of it can be a history of contextual parameters. The context source registry knows whether to address the context storage for a particular context information object or contact the context source directly.

- A context aggregator collects heterogeneous context information, and may provide operations like combining, digitizing and filtering of data.

- A context predictor allows for extrapolating context over time and or space, anticipating current context information and/or historic context. A context predictor can request for complementary context information.

- An ambiguity resolver deals with contradictory and superfluous context and enables certain context information to prevail over the other. It can find the context information that is best suited given all the available contextual data.

- The context service adapter is a special kind of context source that is installed on service deployment. The adapter adjusts generic context information to special purpose needs of application network services. The advantage of this context source is that an application or service can delegate processing to a particular location in the network that is more efficient in terms of resource usage; certainly for applications that run on small devices.

Context sources may be deployed inside the CMF container by the context provider as regular components. Figure 11 shows how context sources are deployed using the context provider. In this Figure we see the CMF container from Figure 10 and a context provider that discovers a context source that produces context information of a certain TYPE. When such a context source has been discovered, either inside the component
repository or at a remote site, the context source is registered with the local context source registry and deployed. Client applications that have subscribed to context information of that TYPE will then start to receive context updates. Client applications can also query specific context information that has been stored by, e.g., a storage context source.

![Diagram showing the deployment of context sources](image)

**Figure 11 Deployment of context sources**

The different types of context sources can be used together in a context sources and managers hierarchy (Dockhorn Costa, Ferreira Pires et al. 2005). Consider the example of the refuel-predictor that suggests the next gas station in the area to users of the car navigation system. Figure 12 shows a possible context sources hierarchy for this refuel-predictor.
The top-level context source is the refuel context predictor. A fuel level wrapper passes on the current fuel level of the car and as soon as the fuel level has reached below a certain value, the refuel context predictor is activated.

It will then retrieve the driving direction of the car from location history storage. Note that the location history storage receives location updates from the location wrapper. The refuel predictor could have used the location wrapper directly; however it would have had to maintain location history for every user independently.

In combination with context information from the road map storage, the context predictor determines on which road and in which direction the car is driving. Then it will finally determine the cheapest reachable gas station by querying gas station location and pricing context information.

Hierarchies of context-sources can be used to infer higher level knowledge from elementary context-types. Using a context sources hierarchy new context sources are formed that can in turn be reused by other applications.

3.4 The AWARENESS Ontology

In the previous we indicated that context sources of the CMF exchange context information using predefined types. These types are defined by the AWARENESS ontology. This ontology is used in a number of ways (Wibbels 2007):

1. The ontology is used to define the syntax and semantics of each piece of context information that is exchanged between a context source and context consumer.

2. The ontology semantically describes types of context that context sources may produce, including proxy context sources. A context type description is used in the discovery of context sources. In this
process, each context source provides a context provider with a type description of the context it produces. Context consumers query the context provider for sources of context types they are interested in.

3. The relationships between users, devices, places etc. contained in the ontology are used to express the relationships between context types.

Only context types that are defined inside the ontology should be exchanged using the CMF. The ontology comprises the context types discussed in section 2.3 and are extended with additional context types when this is required by the emergence of new context.

The current ontology is described using the Web Ontology Language (OWL). OWL is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users (World Wide Web Consortium 2004). These are respectively OWL Lite, OWL DL and OWL Full:

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It provides a quick migration path for thesauri and other taxonomies. OWL Lite also has a lower formal complexity than OWL DL, see the section on OWL Lite in the OWL Reference for further details.

- OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.

- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.
4 The context-aware recommendation service development

In this chapter we discuss the development process and design of a concert recommendation service that is built using CMF context sources. In section 4.1 the application scenario of the service is presented. Section 4.2 gives an overview of the development process that we went through after we defined this application scenario.

The rest of this chapter discusses the design of the concert recommendation service. Section 4.2 structures the requirements for this service and identifies the context sources and services to be developed. Section 4.4 discusses the design of the context sources that we identified. In section 4.5 the concert recommendation service that is built using these context sources is discussed.

4.1 Application scenario

To be able to evaluate the CMF architecture we developed an application scenario for a context-aware service.

The following application scenario has been inspired by the last.fm service. This service announces artists and concerts based on listening habits of the user. This service works as follows: a user that signed up with the service runs a media player plug-in that invokes a web service to update the tracks that have been played by the user. Based on this information the service associates users with artists and concerts.

The recommendation service developed in this chapter filters concert announcements issued by the last.fm service using context information.

Consider the following scenario:

“Sam likes to visit concerts of his favorite artists a lot. Usually he joins some friends that inform him of any nearby gigs. Just recently he started using a calendar application that is said to be context-aware. It notifies him of any concerts that take place in his neighborhood and even take into account his availability (Figure 13).

Whenever Sam is recommended a new concert he receives a brief notification on his mobile device (Figure 13.1) Sam just received a concert recommendation for Friday, June 1st, for a concert of Cake (Figure 13.1).

Since Sam would like to attend the concert by Cake, he bookmarks the concert. The concert is automatically added to his personal agenda. Figure 13.2 shows the personal agenda view. In this view the dates for which a concert has been recommended are marked with a square.

If Sam accesses the detailed view a time scale is shown. On this time scale regular appointments can be scheduled, but it also shows bookmarked concert recommendations. (Figure 13.3) shows the bookmark that he just made.
With the concert recommendation bookmark a number of services are associated that relate to the concert and the performing artist. Sam decides to buy tickets for the concert of June 1st by Cake. In order to buy the tickets he invokes the ticket service from the calendar.”

4.2 Development process

The application scenario that we presented illustrates a practical example of a context-aware application. After analyzing the application scenario we can distinguish a recommendation service that is responsible for issuing concert recommendations and a client application that shows the concert recommendations to the user (Figure 14).

In this chapter we cover the design of the components that use the CMF framework. The
design and usage of the recommendation service by the context-aware calendar application is discussed in chapter 5. It will illustrates how the recommendation service could be used from mobile constrained devices.

Figure 15 shows how we proceeded after defining the application scenario.

**Figure 15 Overview of the development process**

### 4.3 Service requirements

Figure 16 gives an overview of the functionality of the recommendation service described using use cases. A detailed description of the use cases that have been developed for the recommendation service are listed in Appendix A.

**Figure 16 Recommendation service use cases**
The main goal of the recommendation service is to find and deliver concert recommendations to users. Concert recommendations are a subset of the announcements that are issued by the last.fm service, since it only contains a selection of the announcements based on the location and availability of users.

Concert announcements retrieved from the last.fm service (RS02) are filtered using location and availability context (RS08). Location and availability context is updated by the user (RS03, RS04) and stored inside the user profile of the user (RS05).

Before users can use the service they should first register with the recommendation service (RS06). Registration of a new user will create a new user profile for that user (RS05). Since users update their context information using a mobile client application that is wirelessly connected to the service. The user should inform the user of any device address change, otherwise the recommendation service might become unable to deliver concert recommendations to the mobile client (RS07).

4.4 Design of the context sources

The recommendation service is implemented using CMF context sources. We identify three types of context sources:

- The announcement context source is responsible for retrieving a list of concert announcements that selected by the last.fm service for a give user. The announcements should be stored inside the user’s profile.

- The location context source is responsible for storing the location of a user along with the user’s profile. The location context source is accessed by multiple mobile clients that deliver location information.

- The availability context source is responsible for storing availability information along with the user’s profile. The availability context source is also accessed by multiple mobile clients that deliver their availability information.

4.4.1 User profile data model

Context sources associate context information with the central CMF user profile. The model that makes up the user profile currently used by the CMF prototype is shown in Figure 17. A user is identified with its full name and a login name and may own a number of devices. These devices typically have a number of addresses through which they are reachable; depending on the capabilities of the device this may include an IP address, RFID or a Bluetooth device address. Additionally the device’s location is expressed using GPS coordinates.
The user profile that is currently used does not suffice for the recommendation service. Since we also want to keep track of the current availability of the user and concerts announcements the data model of the user profile needs to be extended (Figure 18). Therefore we introduce the announced concert entity which represents a concert that has been announced to the user by the last.fm service. An announced concert is described by the start- and end time of the concert, the performing artist and the location it takes place.

Between users and announcements a many-to-many relationship exists because users may be announced multiple concerts and single announcement may be associated with multiple users. Associations between announcements and users are modeled using an associative class that is used to indicate whether an announced concert is marked a recommendation by our recommendation service.

The agenda entity is an aggregation of appointments planned by the user. An appointment may also represent a concert announcement and therefore it optionally contains a reference to an announced concert.
4.4.2 The announcement context source

Context sources implement either the base wrapper or threaded wrapper abstract class. A threaded wrapper extends the base wrapper with the ability to run as a thread that is launched at a specified interval. The announcement context is implemented using a threaded wrapper because it should iterate over all users of the recommendation service periodically. For each user the last.fm account name is looked-up after which the list of concerts is downloaded. The list of concerts announcements is then parsed and stored along with the user profile. The structure of the announcement context source is shown in Figure 19.

Only unique concert announcements are stored by the context source. Since lists of announcements are downloaded periodically for every user, this requires intelligent handling of announcements. The service needs to determine whether an announcement was already retrieved for another user and if this is the case it should associate the already existing announcement with the user. Incoming announcements are always related using the recommendation association with the recommended field set to false. This field is set by the recommendation service if this announcement meets the criteria for becoming a recommendation.

Handling announcements also includes notifying subscribed services (in this case the recommendation service) about new or changed announcements. An overview of the state transition of the announcement context source is shown in Figure 20.
Figure 20 States of the announcement context source

The announcement context source also supports queries that allow services to select and retrieve announcements synchronously. The query interface of the announcement context source should support the selection of announcements based on start- and end time and location.

4.4.3 The location context source

The location context source handles location context of multiple clients. It extends the base wrapper class and should export a service that is capable of handling client location updates. Figure 21 shows the structure of the location context source.

The location wrapper service is exported an RPC service is invoked by a remote location listener that is installed on a client device. When the RPC service is called the location update is stored into the user profile.

The location wrapper service is also responsible for notifying subscribers of location updates. It does this immediately after a location update has been received by the location wrapper service. A mobile client should invoke the updateContext() method and supply the username and GPS coordinates as parameters.
The location context source also supports the query interface which allows applications to request the current location of a user. An optional extension of this context source is the ability to query past locations of a user but this feature is currently not supported.

4.4.4 The availability context source

The structure of the availability context source is similar to that of the location context source (see Figure 22). Instead of location information the context source maintains a server-side copy of the agenda of a user.

When a client invokes the *updateContext()* method it must supply the username and one or more appointment with a modifier that indicates whether that appointment has been added, changed or deleted. This results in an update of the server-side agenda for that user (see Figure 23).
The availability context source provides a query interface that allows for querying whether a user is available at a specified time interval. Clients should call the `isAvailable()` method and supply the username and the start- and end time to find out if the user is available.

The design of this context source assumes that every user uses a single device to update the calendar. An optional extension of this context source would be to allow multiple devices per user to update the server-side agenda. This is beyond our current scope, since it would add complexity with respect to merging multiple agenda’s.

### 4.5 Service description

The recommendation service implements use case RS08 which supports two scenarios that are implemented using the context sources that we just defined. The first scenario (Figure 24) that triggers the concert recommendation process is the notification of a new concert announcement. The second scenario is initiated by a location update of a user (Figure 25).
4.5.1 Availability-based recommendation

Availability-based recommendation recommends concerts to users depending on their availability information that they have supplied the system with.

After the recommendation service receives a new concert announcement it queries the availability context source to find out if the user is available according its personal agenda.

Figure 26 shows the interactions between the recommendations and the context sources that may lead to concert recommendation.

When the announcement wrapper notifies the recommendation service of a new last.fm announcement. The recommendation service verifies whether the user is available. If the user turns out to be available the concert announcement is recommended to that user.

Availability-based recommendation is only useful in case the user can reach the concert within a reasonable amount of time. Therefore availability-based recommendation is concerned with concert announcements at locations within 30 minutes traveling from the home location of the user.

4.5.2 Location-based recommendation

Location-based recommendation is targeted at making recommendations while a user is e.g. having a holiday in a foreign city. Therefore these recommendations should only be made when travel time to this location is more than 30 minutes. The validity of these recommendations should also be limited to one week.
When the recommendation service receives a location update, the service selects concerts that will take place at that location on the current date and marks the selected announcements as a recommendation.

Figure 27 shows the interactions between the service and the context sources that may lead to the recommendation of one or more concerts.

Figure 27 Making recommendations based on the location of a user

The location wrapper notifies the recommendation service when a user has updated its location. The recommendation service then queries the announcement wrapper to retrieve the concert announcements that take place at that location. The announcement wrapper returns a list of concert announcements. These announcements are marked as recommendation.
5 The context-aware calendar development

This chapter covers the selection of the mobile application platform on the end-user device and discusses the requirements and design of the context-aware calendar application that uses the concert recommendation service proposed in Chapter 4. Section 5.1 discusses the requirements of the context-aware calendar application. Section 5.2 discusses the Java CLDC/MIDP application platform and its properties. Section 5.3 discusses the design of the calendar application.

5.1 Requirements

Figure 28 captures the main use cases of the context-aware calendar application that should be implemented by the mobile client. A detailed description of the use cases is listed in Appendix B.

![Figure 28 Use case diagram of the context-aware calendar application](image)

The context-aware calendar application is responsible for delivering availability and location context to the CMF and handling notifications that are issued by the recommendation service. Before a user can start using the recommendation service, the user should first register with the recommendation service (MC04).

After a user has registered he can start using the recommendation service. A user is allowed to schedule appointments using his personal agenda (MC07). When a user makes changes to the personal agenda, availability changes are sent to the CMF (MC05).
While the calendar application is active, it listens for notifications about concert recommendations. This may include messages about new, changed or removed recommendations (MC01). When the application receives such notifications, it stores them on the client device. Depending on the type of notification the application may show the notification (MC02) to the user. If a notification about a new recommendation arrives, the user can decide to bookmark the concert recommendation (MC03). This will add the concert to the personal agenda of the user.

Finally the calendar application is responsible for forwarding location information to the CMF at regular intervals (MC06).

5.2 The Java ME application platform

5.2.1 General characteristics

Currently, the Java ME platform is the most prevalent application platform for mobile devices. The Java ME platform supports two types of configurations: the Connected Device Configuration (CDC) that targets PDAs, TV set-top boxes and embedded devices, and the Connected Limited Device Configuration (CLDC) that was specifically designed to meet the needs for a Java platform to run on devices like mobile phones with limited memory, processing power and graphical capabilities.

The Java ME platform is a collection of technologies and specifications that are combined to construct a complete Java runtime environment. The Java ME runtime environment is composed of three elements:

- The device configuration (e.g., CDC or CLDC) provides a basic set of libraries and virtual machine capabilities for a broad range of devices;
- A profile (e.g., MIDP or Personal Profile) is a set of APIs that support a narrower range of devices;
- Optional packages that contain a set of technology-specific APIs.

For our context-aware calendar application we selected CLDC platform, since we are particularly interested in how the inherent limitations of these devices could be overcome using CMF services.

5.2.2 The Mobile Information Device Profile

CLDC devices are typically equipped with the Mobile Information Device Profile (MIDP). This provides a complete Java runtime environment consisting of (Sun Microsystems 2003):

- Limited Connected Device User Interface (LCDUI) API for developing user interfaces;
- Record Management Store (RMS) API for storing data persistently on a device;
- Generic Connection Framework (GCF), for networking with a
server or another device.

The included libraries enable developers creating networked wireless services that realize:

- Interactive and flexible user interfaces. The LCDUI API provides high-level components (e.g., lists and forms), which offer standard interactions, and lower-level components e.g. canvas which enable customized look-and-feel.

- Applications that work even when the device is disconnected from the network. These applications may embed UI logic on the client together with the data that drives, possibly using the RMS API to store the data persistently.

- Networked application clients that connect to wireless services using standard networking protocols. MIDP requires all devices to support HTTP which is the same protocol that is used by Web browsers.

The Java CLDC/MIDP environment may be extended with optional APIs (The Java Community Process 2007). Since some of the APIs depend on very specific device capabilities it is not possible to use the optional APIs on every device. For example to be able to retrieve cell or GPS location information a device must support the JSR-179 API and to be able to access the native calendar on a device, it must support JSR-75.

5.2.3 Basic CLDC/MIDP distributed programming model

In (Sun Microsystems 2003) a basic programming model for CLDC applications that use distributed services is proposed. Figure 29 shows the high-level architecture of a wireless Java application that accesses a distributed Java 2 Enterprise Edition (J2EE) application server.

The J2ME client application that is implemented using MIDP is referred to as a **MIDlet**. The MIDlet invokes distributed services by accessing a Java HTTP Servlet on the J2EE application server.

![Figure 29 High-level architecture of a Java Wireless application](image)

When a MIDlet communicates with a Java servlet the following events take place (Sun Microsystems 2003):
The client encodes an application request and packages it in an HTTP request;

- The servlet receives the HTTP request and decodes the application request. The servlet or some delegate (such as an enterprise bean) performs the work specified by the application request;

- The servlet encodes the application response and packages it in an HTTP response;

- The client receives the HTTP response and decodes the application response it contains. The client may instantiate one or more objects and perform some work on these local objects.

The servlet interprets requests from the MIDlet, and in turn, dispatches client requests to Enterprise Java Bean (EJB) components, which encapsulate the application’s business logic. These EJBs reside in an EJB container and provide standard services such as transactions, security, and resource management. When the requests are fulfilled, the servlet generates a response for the MIDlet.

### 5.2.4 MIDP application types

On the client side we selected the MIDP as the application platform for the context-aware calendar application. According to (Violleau and Ortigas 2003) the design of a MIDP application is influenced by three parameters, namely:

- The size of the local data model;
- The local processing that needs to be performed by the device;
- The dependency on a network connection.

The type of a MIDP client application depends on the size of the local data model, whether it distributes or processes program logic and to what extent it depends on network connectivity.

![MIDP Client Application Spectrum](image)

**Figure 30** MIDP client application spectrum

Figure 30 shows the MIDP client application spectrum. Thin applications, are fully distributed. These applications do not depend on a local data store or local processing by the host device. Since these application distribute these responsibilities they fully depend on the network communication with their peer. To the other end of the spectrum there is
the class of standalone applications that do not use any networked resource at all. In between there are the thick wireless clients that might use both local resources as well as distributed resources.

5.3 Design

The design of the context-aware calendar is divided into three parts: the design of the context listeners that publish context information to the CMF, an RPC server listens for notifications about concert recommendations and the calendar GUI that is used to maintain the appointments inside the personal agenda of the user.

5.3.1 The application model

Figure 31 gives an overview of the application model. At the center of this model there is the ContextAwareCalendar MIDlet that initializes the other components.

![Figure 31 Application model of the context-aware calendar](image)

The next steps are executed during the initialization phase of the application:

4. The CalendarStorage is initialized. The CalendarStorage uses the MIDP RMS storage to persist the local data model, discussed in section 5.3.2.

5. The calendar storage has started the AvailabilityListener interface is registered with the CalendarStorage. This interface is invoked when changes are made to the calendar of the user.
6. The `AvailabilityListenerImpl` invokes the `updateContext` RPC service `AvailabilityWrapperService` to inform the CMF when the availability of the user has changed.

7. The location context listener is registered with the JSR-179 `LocationProvider` (The Java Community Process 2007). The `LocationListener` interface is invoked by the `LocationProvider` when new location information becomes available. The `LocationListenerImpl` invokes the `updateContext` RPC service that has been exported by the CMF `LocationWrapperService` (see section 4.4.3).

8. The `NotificationHandler` is started. The `NotificationHandler` opens a server socket connection to listen for notifications coming from the `RecommendationService`.

9. The `CalendarCanvas` is initialized, which represent the user interface. The `CalendarCanvas` shows a browseable per month view of the personal agenda of the user. A date can be selected to browse the calendar detail view realized by the `CalendarDetailCanvas`.

10. The `CalendarDetailCanvas` can be used to add, remove or change appointments. The `CalendarDetailCanvas` uses the `CalendarStorage` class to realize this behavior.

5.3.2 The local data model

The client-side data model is a partial replication of the server-side data model that is used to support processes that are executed on the device. Figure 32 shows the persistent local data model consisting of user information, the personal agenda of the user and the appointments that are contained in this agenda. An appointment may have a reference to an announcement indicating that the appointment is in fact a concert recommendation issued by the recommendation service.

![Figure 32 Local data model](image-url)
The model is a subset of the data model store in the user’s profile shown in Figure 18.

5.3.3 The notification handler

The notification handler should implement the requirements of MC01. It handles notifications about concert recommendations that are issued by the recommendation service. Notifications that have been received by the notification handler are not directly stored into the main data model. First they are saved into notification storage. This is a temporary storage that stores notifications until they are processed by the application.

Notifications might be one of the following types (Table 1):

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTIFY_NEW_REC</td>
<td>A new recommendation</td>
</tr>
<tr>
<td>NOTIFY_CHANGED_REC</td>
<td>A changed recommendation</td>
</tr>
<tr>
<td>NOTIFY_REM_AV</td>
<td>A recommendation that has been removed because the availability of the user has changed</td>
</tr>
<tr>
<td>NOTIFY_REM_LOC</td>
<td>Recommendation has been removed because the user has moved away from the location</td>
</tr>
</tbody>
</table>

Table 1 Supported notification types

Figure 33 shows the structure of a notification. The notification type is kept general to be able to support multiple types of notifications. A notification is described by a unique id, the type of the notification that reflects one of the notification types listed above and the content of the notification. Just after a notification arrives the processed flag of the notification is set to false and the application is signaled that new notifications are ready to be processed.

If the notification is of type NOTIFY_NEW_REC, the notification is shown to the user as specified in use case MC03. The notification can either bookmarked or rejected. If the user decides to bookmark the notification, new Appointment and Announcement instances are created inside the Agenda of the user (Figure 32).

In case a recommendation has been changed with respect to its contents (NOTIFY_CHANGED_REC), the application looks up the Announcement with the associated last_fm_id and updates its contents using the contents of the notification. After the changes has been performed, the notification processed flag is set to true.
In case a recommendation has been removed (NOTIFY_REM_AV, NOTIFY_REM_LOC), both the Appointment and Announcement are removed from the storage. The notification processed flag is set to true.
6 Implementation details

This chapter discusses the main issues that we have dealt with during the development of both the recommendation service and the context-aware calendar.

In section 6.1 the development of the announcement context source is discussed. The process of developing the announcement context source is illustrative for the development of context sources that need to extend the AWARENESS ontology. Section 6.2 shows how Jena and Jastor can be used to persist the data that is exchanged using context sources. Section 6.3 discusses how CMF clients typically connect to the CMF and discusses the current limitations of these CMF utility classes when used by a CLDC client. Section 6.4

6.1 The concert announcement context source

Context information exchanged using context sources should comply with the AWARENESS CMF ontology. The concert announcements are parsed into the structure shown in Figure 34 which is part of the context model shown in Figure 18. To be able to exchange information that matches the structure of this context model, the AWARENESS ontology needs to be extended.

![Figure 34 Announcement structure](image)

For extending the AWARENESS CMF ontology we used the Protégé 3.2.1 ontology editor (Horridge, Knublauch et al. 2004). First of all, we created an OWL ontology file called AnnouncementCS.owl. For our purpose we chose the OWL Lite language since we deal with an ontology that does not require complex constraints.

We chose the ontology URI: http://asna.ewi.utwente.nl/AnnouncementCS.owl, which reflects the namespace of this ontology, and imported the required namespaces:

- http://www.w3.org/2002/12/cal/icaltzd (icaltzd prefix)
http://amigo.gforge.inria.fr/owl/Amigo.owl (Amigo prefix)

The next step has been to determine the possible relationships and constraints on the entities that are required by the context source (Figure 34) and the entities that already exist in the ontology. We first tried to find an entity that matches the structure of the Announcement class. The Announcement class clearly matches the structure of the Vevent class that is specified in the icaltzd ontology (see section 3.4 which discusses the contents of the AWARENESS ontology). Using the Protégé editor we indicated that this class is a subclass of the icaltzd:Vevent class. We also made the Announcement class a subclass of the Amigo:Information class.

Since objects of type Announcement should keep a reference to the original last.fm announcement, we need to add last.fm_id property. Using the Protégé editor either a data type property or an object property can be added. Object properties apply to properties that refers to an object, while data type properties refer to a literal value. In this case we will keep the literal value of the last.fm_id, since this is a remote object that we cannot reference directly. We added the property to the Announcement domain and indicate that it is a literal of type string. We added the User class as a subclass of the Amigo:Person class and add three data type properties of type string respectively fullname, loginname and last.fm_user.

Finally, we modeled the associative Recommendation class using OWL. To this end we used an inverse functional object property and a sub property.

<table>
<thead>
<tr>
<th>OWL Class</th>
<th>Property</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announcement</td>
<td>isAnnouncedTo</td>
<td>i: receivedAnnouncement</td>
</tr>
<tr>
<td></td>
<td>isRecommendedTo</td>
<td>s: isAnnouncedTo</td>
</tr>
<tr>
<td>User</td>
<td>receivedAnnouncement</td>
<td>i: isAnnouncedTo</td>
</tr>
<tr>
<td></td>
<td>isRecommendation</td>
<td>s: isRecommendedTo</td>
</tr>
</tbody>
</table>

s: sub property
i: inverse functional

Table 2 Inverse functional and sub property to model associative Recommendation class
Figure 35 shows a user entity that has received two announcements A and B. Announcement B has also been recommended to this user, since also the inverse functional sub property has been set.

Figure 35 Modeling the associative UML relationship using OWL

The resulting OWL file is shown in Appendix C.

6.2 Generating Jena interfaces using Jastor

We have used Jastor to generate Jena Java Beans from the AnnouncementCS ontology. Jastor generates Java interfaces, implementations, factories, and listeners based on the properties and class hierarchies in a web ontologies (Jastor 2007). We ran Jastor on the AnnouncementCS using a Jastor Ant task (see Appendix D).

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnouncementCS_DOT_owlFactory.java</td>
<td>Factory for instantiating objects for ontology classes in the AnnouncementCS.OWL ontology.</td>
</tr>
<tr>
<td>Announcement.java</td>
<td>Interface for the Announcement ontology class.</td>
</tr>
<tr>
<td>AnnouncementImpl.java</td>
<td>Implementation of Announcement interface.</td>
</tr>
<tr>
<td>AnnouncementListener.java</td>
<td>Implementations of this listener may be registered with instances of User to receive notification when properties changed, added or removed.</td>
</tr>
</tbody>
</table>

Table 3 Overview of the files generated from the AnnouncementCS ontology using Jastor

For the User ontology class similar classes have been generated.
6.3 Communication with the CMF

Java CMF clients can use CMF utility classes to access the functionality of the CMF. The CMF helper class is used to initialize incoming and outgoing connections to the CMF. The CMF clients should first initialize this CMF helper class.

Figure 36 gives an overview of how the CMF client application configures itself using the CMF Helper class. The initialization of the CMF helper class configures the import manager that handles outgoing connection types. By default the import manager is configured to use the CMF RPC importer. After initialization of the import manager, the CMF helper initializes the service registry manager, which allows for the discovery of services inside the CMF containers. By default, the CMF helper adds a service registry connector that connects with a CMF container that runs on the same host as the helper.

![Figure 36 Overview of CMF client application configuring itself using the CMFHelper utility class](image)

After the CMF helper has been used to setup communication with the CMF, the client application configures the default exporter. The default exporter is a socket RPC server that is required to handle CMF responses. The socket RPC server opens an RPC connection and configures the kXML pull parser to handle incoming data from the CMF server. The remotaable data helper is used to parse, serialize and deserialize data that is handled by this parser.

A client application that wants to receive location updates from the CMF server exports its context notification interface using the export manager. Then, after providing the right credentials, it can add a location context subscription using the subscription helpers. If this subscription is successful the client application starts to receive location updates through its exported context notification interface.

The CLDC client that runs the context-aware calendar needs incoming and outgoing connections to the CMF respectively to receive concert recommendations and to supply context information to the CMF context sources, respectively. The utility classes also
seem helpful for connecting to the CMF from the MIDP device. However, there are a few reasons why the utility classes cannot be used in their current form:

- The absence of Java reflection mechanisms on the MIDP device, i.e. the ability to examine or modify the runtime behavior of applications, affects the implementation of the server stub classes.
- The (server) socket classes for MIDP are implemented slightly different from the common java.net.Socket and java.net.ServerSocket classes.
- The restricted implementation of the Vector and HashTable classes lack object cloning methods that are frequently used by the implementation.
- The Thread class lacks the method that allow to set a Thread instance as a so-called deamon thread.
- Types like java.io.File, java.io.PrintWriter and java.util.StringTokenizer are not available under the MIDP environment.

The listed restrictions and incompatibilities of the CMF utility classes with the MIDP platform combined with the impact of the required changes to the current implementation, made us to give up rewriting/using the current utility classes.

6.4 Application-specific client implementation

The context-aware calendar application that we propose in this work illustrates how CMF services could be used from constrained mobile device. We investigated the use of a specific device type, namely Java MIDP devices and the ability of these devices to connect and use services of the CMF through the CMF RPC protocol.

Even though the use of the CMF utility classes show that there are some important differences between the MIDP environment and a regular Java platform, in our experience the CMF RPC mechanism could be used on this type of devices. Although this would require more insight into the CMF RPC protocol.

Our approach consisted of the design and development of a recommendation service and an MIDP client application that accesses the CMF service directly using the CMF RPC interfaces. Current browsing technologies that would be required for implementing the optimal thin client lack the ability process interactive pages, which makes it hard for application developers to build interactive clients that meet the requirements of such thin clients.

To be able to deliver this context information, a platform-specific client is still required to cache and transfer context generated by the mobile client to a context management framework like the CMF.
7 Conclusions

This chapter presents the conclusions of our work and recommendations for future work. Section 7.1 lists the main advantages and disadvantages of using the distributed context management for constrained mobile devices. Section 7.2 highlights the benefits and relevance of context-awareness for mobile browsing technologies. Section 7.3 lists the main factors that influence a possible shift towards the development of thin mobile client applications. Section 7.4 recommends the topics that should be subject to future work.

7.1 Distributed context management

The CMF provides a well-defined structure for the development and deployment of context sources and context-aware services. This helps developers focus on the retrieval of context, because context subscription and discovery mechanisms are readily available. Also the use of the import and export manager is very flexible and facilitates developers to setup communication with applications or other CMF instances using CMF RPC, HTTP or SOAP.

The design of CMF CLDC client application that we propose in this work, uses a distributed CMF server to handle context information gathered by CLDC clients. This design decision has a number of consequences for CLDC applications:

- The context information that is generated needs to be cached on the mobile device, since a connection to the CMF server might not always be available.
- Context acquisition and services are not managed by the CMF container.
- Context-aware services need to be accessed remotely.

Our approach helps to save resources that would otherwise be occupied by CMF services and sophisticated context-processing, however it introduces a number of challenges concerning caching context and using remote service efficiently.

Due to the limitations of CLDC devices, common Java libraries might be incompatible with the CLDC platform and would need to be ported. Using the distributed CMF takes away the need for porting libraries that would otherwise be required on the local device to interpret and reason about context information or to implement services.

7.2 Context information might improve mobile browsing

The use of a context-aware services could improve the way users browse information on constrained mobile devices. Applications could pro-actively select information using context variables that are gathered from the devices environment. Our context-aware calendar illustrates this principle, but also shows that it would require a more flexible client if we would want to retrieve content not related to concerts.

We have illustrated that availability and location information can help to select data
using the recommendation service. These context types are two of the most obvious context types that can be acquired from a mobile device. Other information like call history or presence information could be other suitable context types to be acquired. The possibilities of using information acquired via e.g. Bluetooth into context information is a topic that still needs further investigation.

7.3 Shift in mobile application types

It is likely that more mobile applications will shift in the direction of the thin client paradigm. This would remove the need for replication and synchronization of the data model between a client and the server-side data model and improve the portability of applications among different device platforms. It also makes applications more flexible, since they are fully distributed so that only a single web browser would be required to access these applications.

The shift in CLDC application types, therefore, heavily depends on technology enablers like interactive browsing technologies, the preparedness of network provides to shift to flat-rate pricing for data services and the availability applications that attractive enough for the end-user.

It still remains important to consider whether the availability of the network connection impact the availability of the application. For example, it would not be reasonable to depend on the network connection for accessing the address book that contains the telephone number of a friend that the user wants to call.

In the case of the context-aware calendar applications, local processing and a local data model is required to support acquisition and caching of context. After isolating context processing from the actual user interface that is used to issue recommendations, a thin client for receiving concert recommendations becomes possible.

7.4 Future work

In this work we showed how the AWARENESS CMF facilitates the development of context-aware services by structuring the implementation and deployment of context sources. In its current form, deployment of a CMF container on constrained mobile devices is clearly not an efficient option. It could be considered to develop a lightweight alternative to the CMF that is responsible for capturing context on these devices and make this information available to a distributed context management server.

The context-aware calendar application that we designed and partly developed during this research project consist of a MIDlet application that is specifically aimed at providing concert related services to the end-user. This is not very flexible in case the application needs to be extended with new functionality. More flexible alternatives, that do not require a new application to be installed on the host device when applications change, should be considered.
References


Wibbels, M. (2007). "CMF for Developers, How to write a CMF context source or a CMF client."

APPENDICES
Appendix A – Use cases of the recommendation service

![Recommendation Service Diagram]

**Figure 37 Use case overview of the recommendation service**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| RS01| Handle user context         | **Summary:**                                                                                                                                     Handles any context information that relates to users. This may include storing and reasoning about incoming context information.  
**Pre-conditions:**  
- Incoming context information.  
**Post-conditions:**  
- Context is handled and any possible actions associated with the incoming context information have delegated to the responsible component.  
**Course of events:**  
1. Receive incoming data  
2. Determine the required action to be performed |
| RS02 | Retrieve last.fm concert announcements | **Summary:**  
Periodically retrieves a list of concert announcements that is generated by the last.fm service. Downloading the list of concert announcements requires a user to supply its last.fm username.  
**Pre-conditions:**  
- The last.fm username of the user has been stored inside the user’s profile.  
- Retrieval of the announcements for this user should be executed according to the schedule.  
**Post-conditions:**  
- Announcements are associated with the user’s profile.  
**Triggers:**  
- Scheduler triggers retrieval of announcements for a user  
**Course of events:**  
1. Select last.fm username  
2. Download list of announcements  
3. Parse list of announcements  
4. Store new announcements  
5. Update changed announcements  
6. Associate announcements with the user’s profile  
7. Notify recommendation service of any announcement changes  
**Notes:**  
The retrieved announcements of different users may be the same but should only be stored once. Consequence: different users may reference the same announcement. |
<p>| RS03 | Update user | Summary: |</p>
<table>
<thead>
<tr>
<th>Location</th>
<th><strong>Incoming location information is stored with the user’s profile. Location information is received as geographical coordinates.</strong></th>
</tr>
</thead>
</table>
| **Pre-conditions:** | - Location information should have a valid timestamp  
- Timestamp of the received location is newer than the location that is currently stored inside the user’s profile. |
| **Post-conditions:** | 1. Location information stored inside the user’s profile is updated with more recent location information. |
| **Triggers:** | - User updates location information |
| **Course of events:** | 1. Get timestamp of location information  
2. Get user that supplied the location information  
3. Store location information |

<table>
<thead>
<tr>
<th>RS04</th>
<th>Update user availability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary:</strong></td>
<td>Incoming availability changes are stored with the user’s profile. Availability changes are received as calendaring events that describe an appointment and are supplied with a modifier that indicates whether it is a new, changed or deleted event.</td>
</tr>
</tbody>
</table>
| **Pre-conditions:** | - Availability changes should have a valid timestamp  
- Timestamp of the received availability information is newer than the last availability information that was processed for this user.  
- Availability changes should always be processed in the order in which the changes have been made.  
- Calendaring events should always use unique identifiers. |
| **Post-conditions:** | - Availability information stored inside the user’s profile is updated |
| **Triggers:** | - User updates availability information |
| **Course of events:** | 1. Parse availability change  
2. Determine new/changed/deleted |
In case of a new calendaring event:

3. Determine if calendaring event already exists (exception if it does exist and skip step 4)
4. Add event to user’s calendar that is stored in its user profile.
5. Notify recommendation service that user updated its availability.

In case of a changed or deleted calendaring event:

3. Lookup event with the corresponding identifier (exception if the calendaring event does not exist and skip step 4)
4. Update or delete calendaring event
5. Notify recommendation service that user updated its availability.

<table>
<thead>
<tr>
<th>RS05</th>
<th>Manage user profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary:</strong></td>
<td></td>
</tr>
</tbody>
</table>
Manage user profiles of registered users. The user profile contains information like names, preferences, but also context information that describes the user.

**Pre-conditions:**
- Offered data has a relationship with a registered user.

**Post-conditions:**
- Data is stored and associated with a user.

**Course of events:**
1. Receive data that should be associated with a given user
2. Store the data with that user

**Notes:**
The user profile manager could be extended with context reasoning mechanisms that analyze the user profile data or the relationships between the context of different users.

<table>
<thead>
<tr>
<th>RS06</th>
<th>User registration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary:</strong></td>
<td></td>
</tr>
</tbody>
</table>
Registration of users handles user registration requests and is used to create and remove user profiles.

**Pre-conditions:**
- User registration request
- New registrations need to include last.fm user name

**Post-conditions:**
- Registration request is executed
| RS07 | Update user device address | **Course of events:**  
New registration:  
1. Check existence of last.fm user  
2. Create new user profile  
Remove registration:  
1. Remove user profile  
| **Summary:**  
Handles device address updates of a user.  
**Pre-conditions:**  
- Address of the user’s device changed.  
**Post-conditions:**  
- Device address has been updated inside the user’s profile.  
**Triggers:**  
- Address of the user’s device changed.  
| **Course of events:**  
1. Update device address for this user.  
2. Notify other components of address change.  |
| RS08 | Manage recommendations | **Summary:**  
Manage recommendations involves selecting announcements that could be interesting to the user based on location or availability information. It is also responsible for deselecting announcements that are not interesting for a user any longer.  
**Pre-conditions:**  
- Announced concerts can be visited according to the user’s availability, or,  
- Announced concerts are near the current location of the user.  
**Post-conditions:**  
- Concert announcements that are selected because the user is available or because the user is close to a location where a concert is being held are marked as a recommendation  
- Recommendations that do not match these criteria any longer are removed.  
**Triggers:**  

In case of location based recommendation (RS03):
   - Location update of a user.

In case of availability based recommendation (RS02):
   - A new announcement that has been issued to a user.

**Course of events:**

In case of location based recommendation:
   1. Select announcements based on location
   2. Mark the resulting announcements as recommendation

In case of availability based recommendation:
   1. Determine if the incoming announcement can be attended by the user based on its availability
   2. Mark the concert announcement as recommendation if the announcement can be attended.

**Notes:**
Uses ‘Handle user context’ (RS01).
Appendix B – Use cases of the context-aware calendar

Figure 38 Use case overview of the context-aware calendar application

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| MC01 | Receive notifications       | **Summary:**
|      |                             | Receive notification about concert recommendations.                                                  |
|      |                             | **Pre-conditions:**
|      |                             | - User has registered with the recommendation service.                                                 |
|      |                             | **Post-conditions:**
|      |                             | - Notification are parsed and stored inside the notification storage.                                 |
|      |                             | - Notifications have not been showed to the user yet.                                                   |
|      |                             | **Course of events:**
|      |                             | 1. Receive notification                                                                               |
2. Parse notification  
3. Store in notification storage  

**Extension points:**  
- MC02: Show notification dialog  

**Notes:**  
Notifications may contain concert recommendations, or notifications that relate to recommendations that were previously received by the client. Possible notification types are:  
1. New recommendation  
2. Changed recommendation  
3. Recommendation has been removed because availability of the user has changed  
4. Recommendation has been removed because the user has moved away from the location  
5. New services have been added in relation to the concert (ticket service, merchandising, etc.)  

| MC02 | Show notification dialog | **Summary:**  
Show notification dialog about a new concert recommendation.  

**Pre-conditions:**  
- Notification has not been shown to the user before and requires user attention.  

**Post-conditions:**  
- User took notice of the notification  

**Course of events:**  
1. Show notification dialog  
2. Forward user response  

**Extension points:**  
- MC03: Bookmark recommendation  

| MC03 | Bookmark recommendation | **Summary:**  
Recommendation is bookmarked and added to the context-aware calendar application.  

**Pre-conditions:**  
- User has accepted the dialog to bookmark a
| MC04 | **Summary:**
| Register client with recommendation service.
| **Pre-conditions:**
| - User requests registration
| **Post-conditions:**
| - Recommendation service processes request
| - Recommendation service informs whether registration was successful
| **Course of events:**
| 1. Show notification dialog
| 2. Forward user response
| **Extension points:**
| MC03: Bookmark recommendation

| MC05 | Maintain personal agenda | **Summary:**
| The personal agenda allows a user to keep track of appointments or activities. The personal agenda is a browsable agenda that consist of a per month and a per day view. The detailed per day view can be used to add/change/delete appointments or activities.
| **Pre-conditions:**
| - User accesses per day detail view of the personal agenda.
| **Post-conditions:**

**Recommendation**

**Post-conditions:**
- Recommendation is added to the personal agenda of the user

**Course of events:**
1. Add concert recommendation to the users digital agenda.

**Notes:**
A bookmarked concert recommendation is a special entry inside the user’s agenda, that is aware of any changes with respect to services that relate to that concert or performing artist.

Uses: ‘Maintain personal agenda’ (MC07)
Changes are stored inside the personal agenda

### Course of events:

Adding an appointment:
1. Select start and end time of the appointment
2. Enter subject, location and attendees
3. Save appointment

Deleting an appointment:
1. Select appointment or concert recommendation bookmark
2. Press delete
3. Confirm deletion

Modifying an appointment:
1. Select appointment
2. Modify any of the fields, subject, start time, end time, attendees
3. Confirm modifications
4. Save appointment

A special case is that of adding bookmarks that reference a concert recommendation.
1. Add bookmark similar to adding a regular appointment
2. Add a reference to the last.fm id that is also used by the recommendation service

### Extension points
- MC06: Supply availability context

---

<table>
<thead>
<tr>
<th>MC06</th>
<th>Supply availability context</th>
</tr>
</thead>
</table>

### Summary:
The client supplies availability context to the recommendation service that is generated by the context-aware calendar application.

### Pre-conditions:
- New availability information is available

### Post-conditions:
- Availability information is delivered to the recommendation service.

### Triggers:
| MC07       | Supply location context | Summary:  
The client supplies location context to the recommendation service that is generated the device location provider. |
|------------|-------------------------|--------------------------------------------------|
|            |                         | **Pre-conditions:**  
- New location information is available |
|            |                         | **Post-conditions:**  
- Location information is delivered to the recommendation service |
|            |                         | **Triggers:**  
- Device generates location update |
|            |                         | **Course of events:**  
- Connect to the recommendation service |
|            |                         | - Deliver new location information |

Table 4 Use case descriptions of the context-aware calendar application
Appendix C – AnnouncementCS.owl

<?xml version="1.0"?>
<rdf:RDF
  xmlns:amigo="http://amigo.gforge.inria.fr/owl/Amigo.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://asna.ewi.utwente.nl/AnnouncementCS.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:icaltzd="http://www.w3.org/2002/12/cal/icaltzd#"
 xml:base="http://asna.ewi.utwente.nl/AnnouncementCS.owl">
  <owl:Ontology rdf:about=""/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  Ontology extension for the announcement context source</rdfs:comment>
  <owl:imports rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl"/>
  <owl:imports rdf:resource="http://www.w3.org/2002/12/cal/icaltzd"/>

  <owl:Ontology>
    <owl:Class rdf:ID="User">
      <rdfs:subClassOf rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Person"/>
    </owl:Class>
    <owl:Class rdf:ID="Announcement">
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      </rdfs:comment>
      <rdfs:subClassOf rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Information"/>
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/12/cal/icaltzd#Vevent"/>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="isRecommendation">
      <rdfs:subPropertyOf rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#receivedAnnouncement"/>
      <owl:inverseOf rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#isRecommendedTo"/>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Indicates that the announcement was recommended to the user</rdfs:comment>
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:ID="loginname">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#User"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="last_fm_user_name">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#User"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="last_fm_id">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#Announcement"/>
    </owl:DatatypeProperty>
  </owl:Ontology>
</rdf:RDF>
<owl:DatatypeProperty rdf:ID="fullname">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#User"/>
</owl:DatatypeProperty>

<owl:InverseFunctionalProperty rdf:about="#isRecommendedTo">
  <owl:inverseOf rdf:resource="#isRecommendation"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:InverseFunctionalProperty rdf:ID="isAnnouncedTo"/>
</owl:InverseFunctionalProperty>

<owl:InverseFunctionalProperty rdf:about="#receivedAnnouncement">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >Indicates that a user received this concert announcement</rdfs:comment>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:InverseFunctionalProperty rdf:about="#isAnnouncedTo"/>
</owl:inverseOf>

<owl:InverseFunctionalProperty rdf:about="#isAnnouncedTo">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:inverseOf rdf:resource="#receivedAnnouncement"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >Indicates that the announcement has been announced to the user</rdfs:comment>
  <rdfs:range rdf:resource="#User"/>
  <rdfs:domain rdf:resource="#Announcement"/>
</owl:InverseFunctionalProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365) -->
http://protege.stanford.edu -->
Appendix D – Jastor build task for generating OWL interfaces

<!-- jastor task definition -->
<taskdef name="jastor" classname="com.ibm.adtech.jastor.ant.JastorTask"
        classpathref="libs.classpath" />

<!-- jastor task -->
<target name="buildOntology" description="Builds Awareness Ontology">

<jastor destdir="ont/src"
        generateListeners="true"
        generatePropertyCache="true"
        generateVocabularyOnly="false"
        useEntireURIForIdentifiers="false"
        useStrictTypeChecking="true"
        generateCacheInFactory="true"
        usePackageNamesForRestrictedRanges="false">

<!-- target for announcementCS ontology -->
<ontology generate="true"
        path="src/AnnouncementCS.owl"
        url="http://asna.ewi.utwente.nl/AnnouncementCS.owl"
        javaPackage="nl.utwente.ewi.asna.ontology.announcement" />

<!-- awareness ontology -->
<ontology generate="true"
        path="src/AwarenessAmigo.owl"
        url="http://amigo.gforge.inria.fr/owl/Amigo.owl"
        javaPackage="nl.telin.cmf.impl.rdf.ontology.base" />

<!-- (awareness) calendaring ontology -->
<ontology generate="true"
        path="src/icaltzd.owl"
        url="http://www.w3.org/2002/12/cal/icaltzd"
        javaPackage="nl.telin.cmf.impl.rdf.ontology.ical" />

</jastor>

</target>