Design and study of the software architecture for a context-aware nearcast framework

Date April 27, 2009
Author(s) F.M. Kramer

No part of this publication may be reproduced and/or published by print, photoprint, microfilm or any other means without the previous written consent of TNO.

In case this report was drafted on instructions, the rights and obligations of contracting parties are subject to either the Standard Conditions for Research Instructions given to TNO, or the relevant agreement concluded between the contracting parties. Submitting the report for inspection to parties who have a direct interest is permitted.

© 2006 TNO
# Contents

1 **Introduction** ................................................................................................................... 4  
1.1 Motivation .................................................................................................................... 4  
1.2 Problem statement ....................................................................................................... 5

2 **About TNO** ..................................................................................................................... 6  
2.1 TNO .................................................................................................................................... 6  
2.2 TNO Information and Communication Technology .................................................. 7  
2.3 Context ............................................................................................................................. 8

3 **Nearcasting** ................................................................................................................... 9  
3.1 Nearcasting ................................................................................................................... 9  
3.2 Scenarios ....................................................................................................................... 10

4 **Context-awareness** ..................................................................................................... 12  
4.1 Drivers ......................................................................................................................... 12  
4.2 Definition ..................................................................................................................... 12  
4.3 Context information ..................................................................................................... 13  
4.4 Context retrieval .......................................................................................................... 14  
4.5 Context-aware applications ......................................................................................... 14  
4.6 Conceptual models for context-aware systems ......................................................... 15  
4.7 Conclusions ................................................................................................................... 19

5 **State-of-the-art** ............................................................................................................ 21  
5.1 Commercial systems .................................................................................................. 21  
5.2 Research prototypes .................................................................................................. 22  
5.3 Functionality comparisons ......................................................................................... 23  
5.4 Quality comparison .................................................................................................... 28  
5.5 Conclusions ................................................................................................................... 29

6 **Requirements** ............................................................................................................. 31  
6.1 Nearcast scenario ....................................................................................................... 31  
6.2 Domain model ............................................................................................................. 32  
6.3 Use cases .................................................................................................................... 33  
6.4 Context-aware requirements ..................................................................................... 39  
6.5 Quality requirements ................................................................................................ 41  
6.6 Conclusions ................................................................................................................... 44

7 **Software architecture** ............................................................................................... 45  
7.1 Layered decomposition view ..................................................................................... 46  
7.2 Deployment view ........................................................................................................ 50  
7.3 Client-server view ....................................................................................................... 52  
7.4 Context view ................................................................................................................ 58

8 **Validation** .................................................................................................................. 65  
8.1 Implementation ............................................................................................................ 65  
8.2 Functionality ............................................................................................................... 67  
8.3 Quality .......................................................................................................................... 68  
8.4 Conclusions ................................................................................................................... 79
9 Conclusions........................................................................................................................................ 84

10 Future work.................................................................................................................................... 87

A References....................................................................................................................................... 88
   A.1 Articles ........................................................................................................................................ 88
   A.2 Books......................................................................................................................................... 89
   A.3 Websites ..................................................................................................................................... 89

B Bluetooth......................................................................................................................................... 91

C Interfaces ......................................................................................................................................... 94
   C.1 Interface between nearcast client and nearcast center ......................................................... 94
   C.2 Interface between nearcast server and nearcast center or external components............. 94

D Screenshots nearcast client application......................................................................................... 96

E Nearcast center design issues ......................................................................................................... 97
1 Introduction

Recently, TNO Information and Communication Technology (TNO ICT) defined nearcasting as a form of narrowcasting. Narrowcasting is the delivery or exchange of information via various kinds of media, usually by radio or television, to a narrow audience, an individual or a small group of people at a specific place or area [1,20]. Examples of narrowcasting are digital advertising billboards, advertisement screens in shopping centers or information screens in kiosks, airports or railway stations. Another narrowcasting example is a website only accessible for a specific person or group.

Nearcasting is similar to narrowcasting, to a certain extent. The main difference is that nearcasting involves communication on the spot, i.e. communication between the user and the system takes place at a short distance (in the order of 10 meters). This concept of nearcasting enables content providers to distribute and exchange information to mobile devices at specific locations.

An example nearcasting application is mentioned in [15]. The Dutch airline company KLM has started an experiment at Schiphol Airport in which travelers receive a message via Bluetooth that encourages them to check-in via the internet. Bluetooth is an open standard for wireless connectivity between mobile devices at a short range (see appendix B). The system only sends messages to travelers that have Bluetooth turned on. The message is an animated movie and travelers receive this message only once.

Kameleon Media [14] presented another example application which allows users to download movie trailers, movie facts or movie schedules when they are at the cinema (Figure 1). The user needs to install an application on his mobile device that enables him to use the provided services.

![Figure 1 - Mobile device interacting with a nearcast center installed at the cinema](image)

Chapter 3 further explores and delimits the definition of nearcasting, and some example scenarios are used to illustrate the concept of nearcasting.

1.1 Motivation

This thesis presents the results of research at TNO ICT and is the fulfillment for the Master ‘Software and System Engineering’ in Computer Science at the University of Groningen.

The motivation for this research is twofold. First, TNO ICT is an innovative company that is looking for advances in mobile communication services (a more extensive
description of the organization of TNO and its goals can be found in Chapter 2). The
trend of nearcast systems is a new development within mobile computing and among
others, stimulated by the increase of Bluetooth-enabled mobile phones. TNO is
interested in the developments and effects of innovative nearcasting applications.

Second, software engineering plays an increasingly important role in developing mobile
communication services. As new technologies appear and new services become
available frequently, it is important to create and share design practices in order to
improve software quality and keep engineering efficient.

1.2 Problem statement

This thesis’ goal is to study software architecture for a context-aware nearcast
framework that enables the development of nearcasting services. This leads to the
following problem statement:

*Design and study the software architecture for a context-aware nearcast
framework and validate it through implementing a proof-of-concept.*

This problem statement leads to the following research questions:

1. What are the requirements for a context-aware nearcast framework?
   a. What are opportunities for innovative nearcasting services?
   b. What are the benefits of context-aware features?

2. What is the architectural design of a context-aware nearcast framework?

3. To what extent does the software architecture satisfy the quality requirements?

The answers for these questions are discussed throughout this thesis. Table 1 gives an
overview of the structure of this thesis.

<table>
<thead>
<tr>
<th>Ch.</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Nearcasting</td>
<td>Definition of nearcasting including some scenarios</td>
</tr>
<tr>
<td>4</td>
<td>Context-awareness</td>
<td>Theoretical background of context-awareness</td>
</tr>
<tr>
<td>5</td>
<td>State-of-the-art</td>
<td>Comparison of available nearcast systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Answers research question 1.a.</td>
</tr>
<tr>
<td>6</td>
<td>Requirements</td>
<td>Assessment of functional and quality requirements for a context-aware nearcast framework</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Answers research question 1.b.</td>
</tr>
<tr>
<td>7</td>
<td>Software architecture</td>
<td>Architecture description for a context-aware nearcast framework</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Answers research question 2</td>
</tr>
<tr>
<td>8</td>
<td>Validation</td>
<td>Theoretical and prototype validation of software architecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Answer for research question 3</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions</td>
<td>Summary of research questions with their answers</td>
</tr>
<tr>
<td>10</td>
<td>Future work</td>
<td>Discusses future work for the architecture, prototype and nearcasting</td>
</tr>
</tbody>
</table>
2 About TNO

This chapter describes the organization of TNO [19] and the relation with this thesis. Section 2.1 describes the global organization of TNO, section 2.2 describes the market-oriented business units of the core area TNO Information and Communication Technology and section 2.3 describes the context of this master thesis’ research.

2.1 TNO

In 1930, the Dutch Parliament passed the ‘TNO Act’ that regulates applied scientific research in the Netherlands. TNO was established by law in 1932. The mission of TNO, Netherlands Organization for Applied Scientific Research, is to make scientific knowledge applicable in order to strengthen the innovative capacity of business and government. This mission is put to practice by doing consultancy, contract research, testing and giving out licenses on own inventions. The five core areas of TNO are (Figure 2):

- TNO ‘Quality of Life’ researches concrete solutions to problems encountered by industry and government bodies. TNO develops knowledge for important national and international market clusters such as ‘Agriculture and Nutrition’ and ‘Chemistry and Pharmaceutics’. Additionally TNO is an important partner for the government and public sectors in the area of (health) care and employment issues;

- TNO ‘Defence, Security and Safety’ provides innovative contributions to the advance of comprehensive security and is a strategic partner of the Dutch Ministry of Defense to build up the defense knowledge base. They employ their acquired knowledge for and together with contractors;

- TNO ‘Science and Industry’ carries out research to new products, new materials, faster design, development, and production that is more efficient. The services TNO provides in this area are aimed at industry, government, suppliers and R&D organizations. It’s expertise covers the whole process: design, engineering, planning, processing techniques and the management of the manufacturing process;

- TNO ‘Built Environment and Geosciences’ focuses on the spatial organization and use of space in The Netherlands. Topical issues include the use of the subsurface, mobility, the infrastructure, the renewal of the inner cities, the formation of a network between cities and the administrative implications of this as well as the development of regions, partly in a European context. Renewable energy and energy efficiency, too, are themes that play a role in this core area.

- TNO ‘Information and Communication Technology’ brings together telecom and ICT and helps companies, government bodies and (semi-)public organizations to realize successful innovations in ICT. The next section further describes this core area, as the assignment has taken place within this organization.

The headquarters of TNO are located in Delft and each core area has several offices throughout the Netherlands. TNO has in total more than 5000 employees, holds a
prominent position in international science and develops new knowledge together with universities and the top technology institutes.

Figure 2 - Organization of TNO. The research was an assignment from the User Centered Innovation department.

2.2 TNO Information and Communication Technology

The research for this thesis has taken place in the competence area Information and Communication Technology. As the scope of this area is still very broad, it is split in four market-oriented business units (BU):

- The business unit Corporate focuses on all non-public companies outside the telecom sector and is formed from the departments Business Innovation & Modelling and Future Enterprise Strategies;

- The business unit Public has the public sector as market. The departments Telecom Infrastructure and Services, Network & Information Security and E-Business & E-Government are part of it;

- The business unit Mobile focuses on the mobile market of ICT and is formed from the departments Mobile Information Technology, Mobile Network and User Centred Innovation;

- On the other hand, the business unit Wireline focuses on the wired market of ICT. The departments Broadband and Voice Solutions, Business and Network Optimization, Last Mile and Telephony Solutions and Service Architectures and Security are part of this BU.

The research for this thesis was an assignment of the User Centered Innovation (UCI) department. This department has a team of innovators with knowledge of market, user needs, user interfaces, technologies and service development working together in multidisciplinary teams. Its goal is to create and develop new service concepts. The power of the UCI department lies in determining the communication needs of users and developing service concepts, building prototypes and executing pilots with users, in cooperation with partners and contractors.
2.3  **Context**

Although the work for this thesis was an autonomous project, there are several related research projects within TNO ICT. This section discusses these projects and the connection between these projects and the nearcast system.

2.3.1  **Personal Networking Project 2008**

Freeban’s Personal Network Pilot 2008 (PNP2008) \[10\] aims to develop the concept of Personal Networks and the related technology. A Personal Network interconnects the various private networks of a single user seamlessly, at any time and at any place, even if the user is highly mobile. Such private network can contain a large number of networked devices, such as smart phones, laptops, digital cameras, MP3 players, gaming consoles, personal video recorders, in-car entertainment devices and navigation systems. The project focuses on the interconnectivity between such devices, i.e. sharing content, data, applications and resources.

A nearcast system could be seen as part of a Personal Network: users entering the range of a nearcast system may consider this system as part of their Personal Network in order to be able to connect to it and share content, data, applications or resources. Furthermore, this thesis’ results may help developing prototypes in the context of Personal Networks.

2.3.2  **Farcast**

Farcast [9] is a content repurposing platform. It supports services that let content owners and users create, share and publish multimedia content whenever and wherever, regardless of format or device. Farcast takes care of the compatibility between the originating and target device. An example service is the Farcast Reporter, which enables journalists to publish news text, video, images and audio files via their mobile phone. With the Farcast NewsFacts application, another example service, users are able to subscribe to receive the latest news directly on their mobile phone.

The current Farcast applications use mobile network technologies like UMTS or GPRS to communicate with the Farcast platform. The nearcast system could be used as an extra communication channel. In this scenario, Farcast applications will use the nearcast system to download or upload content, while nearcast’s back-end system communicates with the Farcast platform.

2.3.3  **COOL**

The goal of the COOL project is to research, design and build a framework for context-aware and pro-active applications (de Jong et al., 2006). This framework is able to determine context information that is or might become relevant for providing services. This context information can be retrieved from multiple sensor data sources. Examples of such data sensors include location data (indoor and outdoor), healthcare data (heart rate) and activity data (phone or PC activity and calendar information). The COOL framework is designed for processing, aggregating and distributing this context information.

The relation with the nearcast project is that the conceptual model of the COOL framework may be used when designing the architecture for the nearcast system. Especially the ideas for handling context information may be relevant.
3 Nearcasting

Chapter 1 has introduced the term nearcasting, as defined by TNO ICT. Section 3.1 further discusses nearcasting and section 3.2 presents some scenarios to illustrate the possibilities of nearcast services.

3.1 Nearcasting

The general principle of nearcasting is to provide locally targeted services. Content providers are enabled to deliver or exchange tailored content to or with their customers. This thesis focuses on a specific implementation of nearcasting that involves a nearcast platform. This platform consists of a nearcast server (a back-end server), multiple nearcast centers (sort of access points) and multiple nearcast clients (that run on users’ mobile devices). Service providers can install these nearcast centers anywhere in the public environment: stores, railway stations, airports, conferences, events, markets, integrated in street furniture, but also in offices. This nearcast center enables communication with users via their mobile phone, PDA, laptop or other mobile device using a nearcast technology, a short-range communication technology like WiFi or Bluetooth, which limits the nearcast center’s coverage to a ten to hundred meters range around this nearcast center. In the remainder of this thesis, speaking of nearcasting concerns this specific implementation.

3.1.1 Advantages

Nearcasting does not define a specific wireless communication technology to use. There are several alternative technologies available, e.g. Bluetooth (see appendix B), WiFi or UMTS. This thesis focuses on the use of Bluetooth, but aims for a framework in which this nearcast technology can be easily replaced. Advantages of a Bluetooth-enabled nearcast platform are:

- Nowadays, most mobile phones already have Bluetooth. Only a small number of mobile phones are WiFi-enabled.
- Bluetooth does include push functionality such that the nearcast system will be able to push content to users.
- The costs of Bluetooth are low. Bluetooth chips are cheap and therefore included in many mobile phones nowadays. In addition, Bluetooth communication is free for end-users as there are no operator costs.
- Bandwidth of short-range communication technologies will probably continue to be higher than normal network communication technologies. For example, Bluetooth has a bandwidth rate of 3Mb per second, while UMTS has an average rate of 384kb per second.
- Communication technologies like UMTS and GPRS are not available everywhere. Large buildings and tunnels can be areas with no or small UMTS or GPRS coverage, for example. Nearcast centers could be easily installed in these places.
- Bluetooth’s power consumption is a lot lower than WiFi. However, there is significant difference in power consumption between a mobile phone that continuously uses Bluetooth and one that has Bluetooth turned off.
These are just some reasons that motivate the use of Bluetooth. As this comparison is not complete, future work could perform an extensive comparison of available technologies and determine the best available technology.

### 3.1.2 Connection primitives

This section gives some definitions to create a common terminology and avoid ambiguity and misunderstandings. Connection primitives are basic communication mechanisms. The connection primitives used in this thesis are content push, content pull, content download and content upload (Figure 3). These terms could be easily exchanged: content push can be seen from the user’s perspective and the system’s perspective. From the user’s perspective, it means that the user pushes content to the system. From the system’s perspective, it means that the system can push content to the user. Consequently, the thesis should specify continuously which perspective is in question. To avoid this, this section defines connection primitives to prevent further ambiguity.

![Figure 3 - Connection primitives: different ways of communication between the nearcast system and clients.](image)

When a nearcast center sends content to a device, it is called content push: the center pushes content to the mobile device. Instead of pushing data, the nearcast center could also pull information from nearby devices: content pull. In both previous connection primitives, the nearcast center is the initiator of the communication. However, the user could also request information himself. This is called content download. On the other hand, the user is also able to upload content.

Differences between these connection primitives are the direction of the dataflow (from client to server or vice versa) and the initiator of the communication (the client or server). The definitions of these connection primitives will be used throughout this thesis.

### 3.2 Scenarios

To illustrate the definition of nearcast and to show some example nearcast applications, this section describes some scenarios.

#### 3.2.1 Scenario 1: At the cinema

Alice walks with some friends through the city and runs across a cinema. They might go to the movies that evening, but are not sure which movie to watch. Alice takes her phone and starts the nearcast application, which she retrieved during an earlier visit at another cinema. The application shows what movies are scheduled that evening. Each movie has its own page with more information, schedule times, trailer downloads, comments and rankings. Alice likes Ice Age 2 and persuades her friends by showing the trailer of this movie that she downloaded for free with the nearcast application. She also checks the comments and rankings of other visitors about the movie: everyone is
positive. Alice’ friends are convinced and want to buy tickets. Alice chooses the link to buy tickets showed in the application and orders four tickets for the movie for that evening. She receives the tickets in her mobile phone and shows them to a scanner to get in.

Before the movie starts, Alice and her friends have to wait a while. They take a place in the bar and order some drinks using the nearcast application. The waiter brings the ordered drinks and Alice pays with the nearcast application. Alice and her friends are chatting, but they forget the time. Fortunately, the nearcast application notices that they are not in the movie hall yet, so it alarms. After watching the movie, Alice is very excited. She grabs her phone again, leaves a comment, and gives the Ice Age 2 movie a five-star rating.

3.2.2 Scenario 2: Conference news service
Bob is going to a conference with several colleagues. Before they leave, Bob has downloaded an application from the conference’s website to his mobile phone. This application enables several conference-related services. As soon as Bob arrives and enters the conference hall, the application gets the latest information about the conference including a floor map pushed from the nearcast center that is installed in the hall. Using this application, Bob subscribes to the conference news service that enables all installed nearcast centers to send the latest news to nearby mobile phones. Bob is notified each time there is a new item.

The application helps Bob choosing to which talks he wants to go. Using the application, Bob is able to browse through the presentation abstracts, mark interesting talks and view how these talks are scheduled. When two talks occur at the same time, Bob is notified such that he can choose the most interesting talk.

3.2.3 Scenario 3: Tourist information office
Chris is on holiday. After his arrival, he goes to the local tourist information office. There are many people at the office, so Chris grabs his mobile phone. A poster notifies him that he is able to download an application to his mobile phone. With this application, he is able to browse the available tourist information, including a city plan, information about hotels, restaurants and bars and information about tourist places. The application also contains a schedule with upcoming events in the area. This schedule is updated every time Chris is close to a nearcast center. There are several centers placed throughout the city.
4 Context-awareness

Before studying software architecture for a context-aware nearcast framework, different aspects of context information first need to be analyzed. Section 4.1 describes why the nearcast framework needs to be context-aware. Section 4.2 assesses what context exactly is by presenting a definition of context and context-awareness. Section 4.3 gives some examples of context information to give the reader an idea of what context could be. Section 4.4 discusses different ways to retrieve context information and section 4.5 discusses how context can be used in the nearcast framework by presenting different types of context-aware applications. Finally, section 4.6 presents an overview of different conceptual models for context-aware systems that could be used as part of the architecture for the nearcast framework.

4.1 Drivers

Humans are capable of capturing information of the situation, which they use in their communications with other people or devices. This implicit information enables humans to communicate in an intelligent way, keeping into account the other’s context. The motivation for creating context-aware applications is to give computing devices this ability and increasing the richness of communication between human and device and between devices themselves.

Context aware systems are able to provide the right information at the right time in the right place. Users are not disturbed anymore at the wrong times. For example, a mobile phone that knows that a user is in an important meeting, will automatically switch to a silent mode. However, an emergency phone call, which is obviously more important than a regular meeting, will ignore this silent mode and ring loudly.

A nearcast system can be easily associated with context-aware systems, because such center already knows many things about its context, like its environment or location. The nearcast center is able to scan its environment for nearby users, such that possible users can be located. Further, the nearcast center is placed in an certain environment such that the activity of the users can be easily determined. For example, when the nearcast center is placed in a railway station, the users are probably traveling and when the nearcast center is placed in a tourist information office, users are probably tourists looking for information.

A nearcast system thus offers many possibilities for context-aware services, where context-awareness increases the richness of communication between human and device. Therefore, this thesis includes the study for the benefits for context-aware services.

4.2 Definition

Many researchers have tried to define the concept of context. Some definitions give an enumeration of context information types, for example location, identity, time and identities of nearby people. It seems unfeasible to enumerate all different types of context information because “everything” can be defined to be context. Other definitions have provided synonyms for context, for example, referring to context as the environment or situation. These definitions are too abstract to apply in practice. The last category of context definitions provides topologies that capture all types of context
information. One topology, for example, defines that context consists of three kinds of context information: computing-, user- and physical environment. Such definitions are too specific and not all environments are relevant in certain situations (Dey and Abowd, 2001).

Based on this analysis, Dey and Abowd have created the following definition of context:

(Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task).

To illustrate these definitions, a car navigation system is used. In this situation, the user’s task is to navigate from place A to place B. To make the car navigation system context-aware, it should use context to provide relevant information or present services that are relevant to the user’s task. An example of a context-aware service is to retrieve real-time traffic information from other systems and use this information to adapt the route during the trip if the driver misses an exit for example. The traffic information is in this case the context-information.

The definition of Dey is commonly accepted and much referred to in research literature. Therefore, this thesis uses that definition as starting point for exploring context-awareness in the nearcast framework.

4.3 Context information

As stated in section 4.2, context can be “everything”. This section presents the categorization of context from Dey and Abowd (1999) in order to get an idea of what kinds of information are available and which pieces of context can be useful in the nearcast framework.

This topology categorizes context in primary and secondary context (Figure 4). Primary context comprises of the most important types of context: location, identity, activity and time. These context types are also pointers to other useful context information types. Having the identity of a user, the system can determine the user’s agenda, heartbeat and mood for example provided that this context information is available within the system.

Figure 4 - Categorization of context. Location, identity, time and activity are the four primary context information types, while the lower rectangles contain secondary types of context information.
The pieces of secondary context information can be indexed by primary context. In some situations, multiple pieces of context are required to index a secondary context type. For example, to determine the social situation of an entity, the system needs its location and identity.

This categorization is included to give the reader an idea of what context information is or could be. The nearcast system can use context information from all categories. The primary context information types are most important because secondary context information cannot be found without the primary context. On the other hand, secondary context information can be created by aggregating different types of other context information which results in higher-level information that can be more useful. Thus, using this categorization, no difference can be made in which type of context information is more valuable.

4.4 Context retrieval

Before creating context-aware applications, it is useful to know where and how context can be retrieved. This section analyzes different sources of context information. Contextual information can have different sources and is retrievable in different ways.

1. Sensors can be used to measure certain conditions in the environment. A sensor can be a light intensity sensor, temperature sensor, but also a Bluetooth sensor that measures the nearby devices.
2. The computing equipment (e.g. the mobile device) can be used to determine the state of the software, the user’s activity on the equipment, its configuration and preferences.
3. Information from the outside world can be fetched via the internet.
4. The application can combine several sensors or other types of context-information and use a reasoning model to create a new type of context-information. When the application knows that a user is in a meeting room and there are many people around, then the user is probably in a meeting. Reasoning is sometimes very difficult in contrast with measuring and can cause undesired effects when the application makes wrong estimations. However, reasoning about context-information can result in a more high-level context, which is more useful in many cases (Tuomela, 2003).

Nowadays, many context-aware applications are primarily location-based. Generally, location is relatively easy to measure, using GPS for example, and is an objective measurement: no assumptions have to be made and almost no user input is needed. Besides, location information is useful in many applications. However, the goal of this thesis is to look beyond the use of location information only and to study the use of other types of context information in order to improve services.

4.5 Context-aware applications

After studying the definition of context, examples of context information and in what way context information can be retrieved, this section studies in what ways applications can use this context information. Brown et al. (2000) have identified six different types of context-aware applications:
1. Proactive triggering applications perform a specified action or service based on a change in the user’s context.

2. A second class is the type of applications that streamline human-to-human interaction where these humans use computing devices to assist them in their communication, for example to exchange documents.

3. The third type of applications stores events in a user’s life along with the captured context. All these data can be seen as a large memory, in which the user can retrieve information based on the context-based selection criteria.

4. Setting reminders for future context is a fourth type of context-aware applications. Such application allows one to set a reminder for when he finds himself in a specific context. Setting a reminder such as: ‘When I’m at the grocery store, buy …’, is an example of this application.

5. A fifth category is the optimization of behavioral patterns. Such applications will analyze the user’s behavior and advise better alternatives.

6. The last category is to share experiences. Users may share shopping experience, a set of web links, interesting tourist places, etc.

On the other hand, Dey and Abowd (2001) have identified three different types of context-aware applications.

1. The presentation of information and services to a user, where items relevant to the user are emphasized or made easier to choose. For example, if a user requests the list of movies in a cinema, the application displays upcoming movies or movies that match the user’s profile first.

2. The second type of context-aware applications is the automatic execution of a service. An example of this type of application is a mobile phone automatically switching profiles, based on changes in the phone’s and user’s context, or adjusting the volume or screen backlight’s intensity.

3. The third type of context-aware applications is the tagging of media with context information for later retrieval.

These categorizations help to analyze how current applications can use or future application could use context-aware information. Therefore, both categorizations will be used to analyze what types of context-aware applications the nearcast framework uses and suggest how the framework could use context information in other ways.

### 4.6 Conceptual models for context-aware systems

The aim of this thesis is to study software architecture for a context-aware nearcast system. This section will give an overview of different conceptual models for context-aware systems. Dey et al. (2001) have proposed a conceptual framework based on context widgets that mediate between the application and operating environment. Hong and Landay (2001) describe a service infrastructure approach for context-aware computing and Winograd (2001) presents a conceptual model that is based on a blackboard model. Finally, TNO developed the COOL framework (de Jong et al., 2006). Chapter 7 evaluates these models against the requirements for the nearcast framework to determine which model is most appropriate to use. The next sections further elaborate each model.
4.6.1 Widgets

Dey et al. (2001) have presented the context-aware toolkit, which comprises context widgets, aggregators, interpreters, services and discoverers (Figure 5). This section describes each of these components.

Figure 5 - Example configuration of Context Toolkit components, from (Dey et al., 2001). Two external applications use the same aggregator and one of the applications uses an interpreter to translate context information from the aggregator into more useful information. The aggregator uses data from two context widgets that both represent a type of context information based on the corresponding sensor. One of the two widgets is able to execute a context-aware service. The discoverer has no connections because it is accessible by all other components.

- **Context widgets** each represent a specific kind of context information and provide other applications access to this context information, such that the complexity of the sensors that determine this information is hidden for these external applications. Context widgets are mediators between applications and the operating environment and provide reusable and customizable building blocks: a number of context widgets can be used to form a new context widget.

- The combination of several widgets into one can be achieved with an **aggregator**. Developers are able to retrieve several context information entities and combine these into one component. Aggregators simplify applications, because instead of querying different context widgets and calculating the result, the application suffices with querying the aggregator.

- **Interpreters** enable applications to request a specific piece of context information at a specific level of abstraction. The interpreter is able to translate the location information sensed by a location context widget to higher-level location information. For example, if a location widget has determined the GPS coordinates for a user, the interpreter can translate this information into the location of a building or room, which can be more useful to the application. The aggregator has the same behavior as a context widget, such that it is able to notify the application about context changes, it can be queried or polled for updates for example.

- The fourth component in the context-aware toolkit is the **services** component. Applications can let this component execute context-aware actions, such that building blocks can be developed for common context-aware services.
The final component, the **discoverer**, is responsible for maintaining a registry of what capabilities exist in the system. Once context components become available, they register their identity and capabilities at the discoverer. This component is thus aware of what widgets, aggregators, interpreters and services are available.

### 4.6.2 Service infrastructure

Hong and Landay (2001) focus on a service infrastructure to create an environment for context-aware applications. The resulting architecture focuses on privacy sensitive applications, where data is stored as much as possible at the end user’s device to give users access control to their personal information. Context information is stored in so-called infospaces, which are network-addressable logical storage units and represented by clouds (Figure 6). These clouds contain individual types of context information, represented by the small squares. Infospaces reside on infospace servers, represented by rounded rectangles.

![Figure 6 - Example configuration of infospaces, from (Hong and Landay, 2004). Alice’s context is stored in Alice’s infospace, which includes her location, activity and health. The location references to another infospace, the Room 525’s infospace. The context of this room consists of persons and devices. It includes a link to Alice again and to a PDA’s infospace. The infospace of the PDA is stored on a different infospace server and consists of the owner of the device, which is Alice in this case.](image)

Hong and Landay identify several advantages for their service infrastructure approach. Although some of them also apply for the other models, the advantages are described here. First, this infrastructure is independent from the hardware, operating system and programming languages of the devices. Second, the infrastructure improves the capabilities for maintenance and evolution. New devices can be added dynamically to the infrastructure, thus without modifying the rest of the system. The device only needs to know the data format and protocol. A third benefit of a service infrastructure is the sharing of sensors, processing power, data and services. In this way, devices may use the capabilities, sensors, processing power or data, of other devices, decreasing the requirements for the device itself. In addition, by reusing services applications are easier to develop.

### 4.6.3 Blackboard

The blackboard pattern is a well-known architectural pattern that can be used in context-aware systems. All sensors or processes store context information on a central server and services are able to subscribe to changes in specific pieces of context information. Consequently, all communication is going through the central server, which facilitates the addressing and routing of messages.
Winograd (2001) proposes the interactive workspace model, which is based on a blackboard architecture style with two levels of data: the first level blackboard is the event heap. Processes can insert events in the event heap, which distributes these events again to subscribed processes. The events in the heap persist for at most an interaction session and exist only for a short period. The second level blackboard is the context memory, an XML-structured database that allows any process to store and retrieve XML-encoded data.

Figure 7 - Example configuration of the interactive workspace, from (Winograd, 2001). Active badge sensors measure users’ location, which are passed to the Location Observer. The Location Observer passes these location updates to the event heap. Dotted areas show messages that are passed to and from the blackboard. Data flows in the arrow’s direction.

Figure 7 shows the configuration of the interactive workspace architecture for the active badge call-forwarding application. This classical example of a context-aware application administrates the location of employees, who all carry a so-called active badge that can be detected by sensors, and forwards calls to the nearest phone. Events, generated when an active badge enters or leaves a specific space, are sent to the location observer, which in turn posts in the event heap. The active badge application maintains the information in the event heap and is able to subscribe to new events. The application can use the context memory for information that is relatively more static, in this example among others the association between a badge identification number and a person.

4.6.4 COOL Framework

TNO ICT has developed the COOL framework that enables the development of context-aware and proactive applications. Figure 8 shows an overview of the COOL framework, which has some similar concepts as the Context Toolkit. COOL’s context components can be categorized in connector components, aggregator components and computational components. Incoming sensor data is handled by connector components, which can be compared with the context widgets from the Context Toolkit. The same applies for the component aggregator, which is capable of aggregating context information from different sources into one type of context information. Another comparable component is the Object Repository, which is similar to the Context Toolkit’s Discoverer: it holds information about the available components and their capabilities. Using the Service Composer, administrators are able to configure all components in the network.
Figure 8 - Overview of the COOL framework, from (de Jong et al., 2006). Sensor data flows from the sensor components into the connector components of the COOL framework. The red arrows indicate data flowing between the COOL components, while the blue arrows indicate data flowing between the COOL control components and or a COOL component.

The COOL framework is designed according to a number of design principles. First, to enable communication between components, as many different protocols as possible are supported. Second, to support the different requirements and needs of the COOL services, components in the framework are supposed to be loosely coupled. Third, components must implement a minimal set of standardized interfaces within the framework using the so-called COOL Contracts. Finally, components present their data in such way that other components can understand it (semantic integration).

4.7 Conclusions

Because a nearcast center is automatically location-aware and is expected to offer many possibilities for context-aware services, this thesis studies how the nearcast system should use context information and what the benefits of context-aware services are.

Dey’s context definition seems to be commonly accepted in the scientific world and therefore this thesis uses this definition as starting point for exploring context-awareness in the nearcast framework.

Although there are many different topologies to categorize context information, this thesis uses the categorization into primary and secondary context information, because it is a clear and unambiguous categorization. It can be used to explore what context information the nearcast framework can use.

Once it is clear what context information the nearcast framework can use, it should be analyzed how this information can be used. Two classifications of context-aware applications show that there are many different ways to use or apply context information. Chapter 6 will analyze what and how context information should be used.

Finally, this chapter shows that there are different ways how systems can handle context information by presenting four models for context-aware architectures. These models
provide insight in how context-aware systems deal with context information and will be analyzed in Chapter 7 when designing the software architecture for the nearcast framework.
5 State-of-the-art

This chapter presents an overview of existing nearcast systems. Section 5.1 presents existing commercially available systems. Next, section 5.2 introduces some research prototypes. A distinction is made between commercial systems and research prototypes, because commercial systems do not have all information available needed for good comparison, research prototypes on the other hand present some design consideration that can be used in the architectural design. Section 5.3 compares the presented systems on various points of functionality, while section 5.4 focuses on the comparison of quality attributes. Finally, section 5.5 presents the conclusions of this chapter.

5.1 Commercial systems

The following systems form a representative selection of current commercially available systems. This section will describe each of the commercial systems shortly.

1. Bluecasting;
2. M2Blue;
3. Kameleon Media;
4. FuturLink;
5. WideRay Proximity Services.

5.1.1 Bluecasting
Bluecasting is a “proximity marketing system” that focuses on distributing multimedia content to mobile phones [7]. Users, in a radius of up to a hundred meter around the Bluecasting server, are asked to make their Bluetooth-equipped mobile phone discoverable, usually via a poster or screen. As soon as the Bluecasting server discovers a mobile device, it sends the multimedia content to the device via Bluetooth.

5.1.2 M2Blue
In fact, the M2Blue system [18] has the same functionality as Bluecasting. M2Blue distributes mobile flyers (slideshows, possibly enriched with audio) to nearby mobile devices. The flyer may contain a message like ‘SMS SERVICE ON to 4411’. By sending a SMS to the specified number, users are able to subscribe the service. The way users receive new content depends on their position: users nearby an M2Blue access point receive new content via Bluetooth. Otherwise, content is distributed via WAP push.

5.1.3 Kameleon Media
Kameleon Media’s smart tag networks let users communicate with objects in the environment [14]. There are two different solutions: MobiPoint ID and MobiPoint Memory Zone. Both solutions require users to install the Kameleon application, which is available via SMS and WAP Push, the Kameleon website, or the MobiPoint Memory Zone itself.

The MobiPoint ID can be seen as an alternative for WAP Push. When a user is close to a MobiPoint, a web link is sent via Bluetooth to the user’s mobile device that retrieves content from the internet via GPRS or UMTS. Thus, objects in the environment can be associated with a relevant web link, such that users do not have to enter a long URL address. Further communication goes via GPRS or UMTS.
The other solution is the MobiPoint Memory Zone: a kind of content mini-server. Users having the application running are able to download content from the MobiPoint via Bluetooth. An interesting feature is that the system adapts the language of the content.

5.1.4 FuturLink
FuturLink has developed the Wilico Access Point, which is a proximity marketing system [11]. FuturLink presents several examples of how its system may be applied: from hotels and museums to music events and public transport. The system requires the user to install an application on his mobile phone. This application enables Bluetooth communication with the access points, such that the user may browse and download content from the access point.

5.1.5 WideRay Proximity Services
The WideRay Proximity Services framework [22] provides on-location distribution of digital content, applications, and services to mobile devices via short-range wireless technologies (Bluetooth, Infrared and WiFi). The distribution servers are remotely controlled by a globally available service management system. This system has a client application that lets users browse and download available content. This application is distributed to users at their first time meeting with the service point.

5.2 Research prototypes
This section will introduce some research prototypes.

1. Mobile Kärpät;
2. B-MAD;
3. Hermes Photo Display;
4. Location Based Indoor Advertisement.

5.2.1 Mobile Kärpät
Ojala et al. (2004) present a case study demonstrating complementary distribution of static and dynamic multimedia content with TeliaSonera’s iJack service, in and around the arena of a Finnish ice hockey club. Users are able to browse and download static content about the ice hockey club (information uploaded to the service points before the match) from the service point via Bluetooth. Dynamic content (match statistics, video replays) can only be downloaded via GPRS (for mobile phones) or WiFi (for PDA’s).

5.2.2 B-MAD
Aalto et al. (2004) present the B-MAD (Bluetooth Mobile Advertising) system, which consists of a Bluetooth Sensor, Ad Server and a Push Sender. The Bluetooth Sensor discovers nearby devices and sends the addresses of these Bluetooth addresses together with a location identifier over a WAP connection to the Ad Server. The Ad Server maps the address to a phone number, that is supplied when the user registered with the mobile advertisement service, and checks if there are any advertisements associated with the location, that have not been delivered to the end user. The Push Sender subsequently delivers these advertisements to the mobile device by sending a WAP push message. Thus, the B-MAD system uses Bluetooth for discovering nearby devices only.
5.2.3 Hermes Photo Display

Cheverst et al. (2005) have studied the development of the Hermes Photo Display that enables users to send photos from their mobile phone to this display and receive photos from the display on their mobile phone. To send a photo from the mobile device to the display, users should use the native phone application to send photos via Bluetooth to another device. Users have to select the Bluetooth device associated with the display manually. To receive a photo from the display, users should select a photo on the display using its touch screen and select the Bluetooth name of their own device as the recipient.

5.2.4 Location Based Information/Advertising

The Bluetooth Location Based System is a location based information and advertising system that sends simple text messages to mobile devices (Rashid et al., 2005). Two different applications of this system are discussed. The first application is the system sending the latest information on products and discounts to customers of a supermarket. In addition, the system is able to pick a random device that receives discount or gets a free item. The other application is a guided tour: tourists receive location-specific information. This system can also be used to determine what routes tourists walk.

5.3 Functionality comparisons

This section compares the existing systems on a number of issues.

- The connection primitives (from section 3.1.2) are compared because these form the foundation for the rest of the system. The supported services of a nearcast system depend on which connection primitives are available.
- The supported content is compared because the system depends on what kind of content it can exchange with its users and without content there is nothing to nearcast.
- The registration methods are compared because these methods determine how users become aware of the nearcast framework. It is important to remove all barriers for users to get started with using the nearcast framework.
- The application distribution is compared because this is also part of how users can get started with using the nearcast framework.
- The supported context information is compared because this thesis focuses on a context-aware nearcast system. Context information enables user tailored services.

5.3.1 Connection primitives

Table 2 shows the comparison that is made for the support for these connection primitives.

<table>
<thead>
<tr>
<th>System</th>
<th>Content push</th>
<th>Content pull</th>
<th>Content download</th>
<th>Content upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluecasting</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>M2Blue</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Kameleon</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>FuturLink</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>WideRay</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>
The table shows that none of the systems does support all connection primitives. Half of
the systems implement content push and content download, while almost none of the
systems implement content pull and content upload. Most of them only support one
connection primitive, which is either content push or content download. Both these
connection primitives are responsible for data flowing from server to client.
The content pull and content upload primitives are responsible for data flowing from
client to server. Although these connection primitives are poorly supported, they are
promising to provide opportunities for new services. With these connection primitives,
users can upload and publish self-created content, have more control about the content
that is downloaded and services can be made more tailored.

5.3.2 Content
Table 3 shows the results of the comparison of supported
content. The majority of the commercial systems promote on their websites that they
support transportation of most kinds of content mentioned in Table 3.
Some of these systems present no real examples, but the use of Bluetooth does not
really restrict to a specific kind of content. These systems have implemented the object
exchange protocol OBEX (see appendix B) to send all kinds of objects. Actually, the
most important restrictions on content are the mobile device’s capabilities. Newer
mobile phones do support audio and video formats and maybe even Word, Excel and
PowerPoint files, but older phones may not even support HTML pages.

Table 3 - Overview of used content. Key: √ = support, × = no support, - = n/a.

<table>
<thead>
<tr>
<th></th>
<th>Text files</th>
<th>HTML files</th>
<th>Images</th>
<th>Audio</th>
<th>Video</th>
<th>Business card</th>
<th>Calendar events</th>
<th>Applications</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluecasting</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>M2Blue</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Kameleon</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>x</td>
</tr>
<tr>
<td>FuturLink</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>WideRay</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>x</td>
</tr>
<tr>
<td>Mobile Kärpät</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
</tr>
<tr>
<td>B-MAD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hermes</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>LBIA</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

According to the fact that most systems use the object exchange protocol OBEX, which
handles file transfer, and that this protocol has no built-in streaming capabilities, the
current systems are not capable of delivering streaming video or audio. To implement
streaming media, the systems should probably support other Bluetooth profiles, like the
Advanced Audio Distribution Profile or the Video Distribution Profile. Streaming media could be an extension of the functionality.

Although the commercial systems do support business cards and calendar events, there are no real applications identified that use these types of content. The rest of the comparison provides no other remarkable characteristics. Thus, regarding to the type of content the nearcast framework is able to distribute, streaming media could be an opportunity to provide new functionality.

5.3.3 Registration method

Users are able to register with the systems or content providers in different ways. Some systems actively send messages to nearby devices (opt-out), some of them wait passively for clients to serve (opt-in) and some support both (Table 4).

Table 4 - Registration method. Key: ✓ = support, × = no support, - = n/a.

<table>
<thead>
<tr>
<th></th>
<th>Opt-in</th>
<th>Opt-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluecasting</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>M2Blue</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Kameleon</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>FuturLink</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>WideRay</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Mobile Kärpät</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>B-MAD</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Hermes</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>LBIA</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

The first way is called opt-out. Users are registered automatically and without any subscription actions, they receive content. Some people call this mobile spam. In the EU, the law prohibits this way of marketing in the public environment, although in shops, restaurants and other closed environments it is allowed (EU 2000/0189 (COD) article 9).

Opt-in, the second way, means that users register to a content provider and subscribe to receive information on their mobile device. Users are able to indicate that they are willing to receive information either implicitly, by installing a mobile application on his device, or explicitly by submitting some information (a dummy contact or business card for instance) to the nearcast center via Bluetooth. This enables the user to subscribe to a news service that sends new items every time the user is close to the center, for example. In addition, some people are of the opinion that users, who accept an unsolicited message from a nearcast center, implicitly indicate that they are willing to receive more information from this center. In the first place, this could be more user-friendly because instead of creating a dummy contact and sending it to a nearcast center, users only have to decline or accept a message, which is easier and takes less user interactions. However, when users receive too many of these kind of messages, they may become irritated and turn off their Bluetooth.

Although the opt-out method is more user-friendly in first instance, eventually this may lead to problems. Besides, the opt-in methods may be too complex. Therefore, a method to opt-in easily for a nearcast service may be studied.
5.3.4 Application provisioning

Users, not having a client application installed, are not able to interact easily with the system. A possible scenario for a user who does not have a client application installed is as follows: the nearcast center may send a file to the user and the user receives this file. To give feedback or request another file, the user replies by creating a note on his phone and sending this note back to the nearcast server. Such scenario is not user-friendly and therefore not a desired situation.

Table 5 - Application provisioning. Key: \(\checkmark\) = support, \(\times\) = no support, - = n/a.

<table>
<thead>
<tr>
<th></th>
<th>Client application</th>
<th>1</th>
<th>2</th>
<th>3.a</th>
<th>3.b</th>
<th>3.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluecasting</td>
<td>(\times)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>M2Blue</td>
<td>(\times)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Kameleon</td>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
</tr>
<tr>
<td>FuturLink</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
</tr>
<tr>
<td>WideRay</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\checkmark)</td>
</tr>
<tr>
<td>Mobile Kärpät</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
<td>(\checkmark)</td>
<td>(\times)</td>
<td>(\times)</td>
</tr>
<tr>
<td>B-MAD</td>
<td>(\times)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hermes</td>
<td>(\times)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LBIA</td>
<td>(\times)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

If the client has an application installed, these kinds of interactions can be made user-friendly. The problem is, however, how to get the application onto the phone, i.e. how application provisioning is arranged. Table 5 presents an overview of which systems require a client application and how this application is distributed. There are several ways:

1. The first way lets the user send a SMS to a specific number, after which he receives a WAP link via WAP push. The disadvantage of this way is that the user should have installed an access point correctly, i.e. he should have access to the internet to be able to request the WAP link. Also, the user probably must pay for downloading the application and the downloading process takes longer than when the user downloads the application locally.
2. When the user knows beforehand that he will use a service that requires an application, he can download the application from the internet to his PC and transfer it to his phone, e.g. via Bluetooth or a serial cable. The drawback of this method is that the user must install the client in advance.
3. Maybe, the most obvious way is to send the application using the nearcast server itself. There are several alternatives for such scenario:
   a. The nearcast center sends the application to all unknown nearby mobile devices. The server keeps record of devices that have the application installed already and does not send the application again to these devices.
   b. The nearcast center sends a message to the user (in HTML for example) and informs the user about the possible services he can use. The message contains an internet link with which the user can request the application. The application is sent to the user via the nearcast server.
   c. The user explicitly indicates that he wants to receive an application, for example by sending a dummy contact to the nearcast server or choosing his own device on a touch screen that is connected and associated with the nearcast server.
An important part in application provisioning is to determine the manufacturer and model number of the device the application should be sent to, which is discussed in section 5.3.5. Knowing this information, the nearcast system is able to determine the software version that should be sent.

There is no best way to distribute applications, thus further research can be done for improvements of these methods.

5.3.5 Device recognition

As mentioned in the previous section, another issue is how the nearcast server knows what version of the client software it has to send. This depends on the manufacturer and model number of the mobile device. Actually, the nearcast server cannot determine this information using Bluetooth only. With Bluetooth, one can determine the unique Bluetooth address, the Bluetooth name, the class of device (computer, laptop, mobile phone, etc) and the supported Bluetooth profiles.

Herfurt and Mulliner (2004) present a possible solution. Their Blueprinting - Bluetooth fingerprinting - method uses the Bluetooth address and the behavior of the Service Discovery Protocol (SDP) server to identify the manufacturer and model number of a device. Each Bluetooth device has a unique 48-bit identifier of the form MM:MM:MM:NN:NN:NN. The first part of this address refers to the manufacturer or vendor of the device [5]. The behavior of the SDP server can be used to refine further the identification: determine the model number and firmware.

Inquiring the SDP server results in the SDP records that contain the services the device offers and how to use these services. Each service contains a Service Record Handle field, a unique 32-bit number identifier that is registered during device startup. This number is hard-coded in the firmware and thus unique for each model number and firmware. Beside the Service Record Handle, the SDP server also returns a RFCOMM channel or L2CAP PSM number for each service that can be accessed to use that service. The hash value is the sum of all products of the Record Handle and channel number of each service. Combined with the first part of the Bluetooth address, it results in a fingerprint. An example fingerprint is 00:60:57@2621543 for the Nokia 6310. This fingerprint can be looked up in a database that contains values for all existing Bluetooth devices. One drawback of this system is that if a mobile phone runs an application that publishes its own defined Bluetooth service, this method will not work because the SDP record is modified too.

Some of the commercial systems seem to have a method to determine what type of devices they discover, but it is unclear what methods these systems use. When nearcast centers know what kind of devices are close, they are able to behave more autonomously. Therefore, the method of Herfurt and Mulliner should be studied further and if there are any other ways to retrieve more information about remote devices.

5.3.6 Context information

Table 6 presents an overview of which types of context information are used in the different compared systems.

Four types of context information are used hardly or not: consumption time, creation time, social situation and communication channel. These types of context information
could be opportunities for innovative services. What kind of new services could be created by using this information or how existing services could be improved is still unclear.

Location information and device transaction history are used in almost all systems. Because of the nature of Bluetooth, the use of location is rather obvious. The use of the transaction history of devices is also not very surprisingly. In many cases content providers do not want to deliver a message twice to a device. Delivering many times the same message would become very annoying. Therefore, it is useful to keep record of what messages have been sent to a device.

Table 6 - Use of context-information. Key: ✓ = support, ✗ = no support, ? = unknown.

<table>
<thead>
<tr>
<th></th>
<th>Location</th>
<th>Consumption time</th>
<th>Creation time</th>
<th>Device transaction history</th>
<th>Device information</th>
<th>Social situation</th>
<th>Communication channel</th>
<th>Content transaction history</th>
<th>Other nearcast centers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluecasting</td>
<td>✓ ✓ ✗ ✓ ?</td>
<td>✓ × ✓ ✓ ?</td>
<td>× ✓ × ✓</td>
<td>✗ ✗ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>M2Blue</td>
<td>✓ ? ✓ ?</td>
<td>×✓ ✓ ✓</td>
<td>✓ × ✓ ✓</td>
<td>✓ ✓ ✗ ✓</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>Kameleon</td>
<td>✓ ✓</td>
<td>✗ × ✓ ✓ ?</td>
<td>✓ × ✓</td>
<td>× × ✗ ?</td>
<td>× ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>FuturLink</td>
<td>✓ ? ? ?</td>
<td>✓ × ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>WideRay</td>
<td>✓ ✗ ✗ ✗</td>
<td>✓ × × ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>Mobile Kärpät</td>
<td>✓ ✗</td>
<td>✗ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>B-MAD</td>
<td>✓ ✓ ✗ ✗</td>
<td>✓ × ✓ ✓</td>
<td>✗ ✗ ✗ ✗</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>Hermes</td>
<td>✗ ✗ ✗ ✗</td>
<td>✗ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
<tr>
<td>LBIA</td>
<td>✓ ✗</td>
<td>✗ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✗ ?</td>
<td>✓ ✓ ✗ ✗ ✗</td>
<td>✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓</td>
<td>✗ ✓</td>
<td>✗ ✓</td>
</tr>
</tbody>
</table>

The use of device information is used in the FuturLink and WideRay systems. The first offers the content provider the functionality to assign content to a specific device group. The content provider may filter for mobile brand, model, Java support, operating system, display size, video support, audio support, image support or combinations of these. The WideRay system has Intelligent Client Delivery and Management that makes sure that the right application is provisioned to the mobile client. How these systems work exactly is unclear, but they do have mechanisms to determine the remote device’s characteristics.

5.4 Quality comparison

It would also be important to compare the quality attributes of the reviewed systems, because these qualities make systems more competitive. For designing the nearcast framework’s software architecture, it is useful to know what level of quality the reviewed systems have.

To perform a quality comparison, a list of quality scenarios needs to be generated. The systems should execute all these scenarios such that the qualities of these systems can be measured. Unfortunately, it was not possible to execute scenarios on the reviewed
systems, because these systems were not available and only little information about these scenarios was available in literature.

Therefore, this thesis does not include an extensive quality comparison. This results in some limitations for this thesis. It is more difficult to draw quality requirements because there are no numbers available already, and the software architecture evaluation cannot make statements about the nearcast architecture being better or worse than the reviewed systems, with respect to the quality attributes.

5.5 Conclusions

The advances in mobile devices are a primary driver for the rapid development in the area of nearcast systems. Almost everyone does have a mobile phone nowadays and the majority is equipped with Bluetooth, while newer models are equipped with both Bluetooth and WiFi. Now and then, media are reporting experiments or practical applications of nearcasting systems and more and more companies present nearcasting solutions, for a wide variety of applications.

The goal of this chapter was to find opportunities for innovative nearcasting services or improvements for current services by comparing existing nearcasting systems. Several observations have been made:

1. Based on the comparison for support for the connection primitives, services may be developed that also use content upload or content pull, next to content download and content push. Thus, content providers should not only distribute content to end-users, but end-users should also be able to upload content to the system or use the system in a more interactive way. Pulling information from the mobile devices enables the nearcast system to retrieve more context information about users provided that they have installed an application. This context information can be used to provide new services or deliver more tailored services.

2. The content analysis shows that streaming media could be a benefit for nearcasting systems. It is clear that streaming a movie is preferable above downloading it: users can immediately start watching a movie and do not have to wait until the complete movie is downloaded. Especially with longer movies, this will be a significant improvement.

3. Methods for registration and application provisioning could be optimized further. Registration includes the possibility for users to subscribe to a service, such that they are able to receive news updates automatically.

4. The use of other types of context information may lead to new services or improvements of current services. However, such context information is very specific for different events. Therefore, a general way to handle context information should be introduced.

The software architecture should focus on the first and last observations: the enabling of content uploading and content pull and the use of context information. The second and third observations are not studied further to delimit the scope of this thesis. The software architecture should include methods for registration and application
provisioning though, but no extensive study is made to explore other possibilities for these functionalities.

Unfortunately, an extensive quality comparison is not executed because it was not possible to execute quality scenarios on the reviewed systems. This results in some limitations for this thesis. It is more difficult to draw quality requirements because there are no numbers available already, and the software architecture evaluation cannot make statements about the nearcast architecture being better or worse than the reviewed systems, with respect to the quality attributes.

The trends above have been translated into the requirements for the nearcast framework, which are discussed in the next chapter.
6 Requirements

This chapter assesses the requirements for the nearcast framework. Section 6.1 describes a typical scenario to give the reader a quick overview of the required capabilities of the nearcast framework. Section 6.2 describes the nearcast domain model and section 6.3 elaborates the use cases to capture the intended behavior of the nearcast framework. Section 6.4 discusses the context-awareness requirements, section 6.5 presents the quality attribute requirements and finally, section 6.6 concludes this chapter.

6.1 Nearcast scenario

This section describes a typical scenario for which the nearcast framework could be used. Some functionality already exists in other systems, but according to the conclusions of the previous chapter, also some new functionality is included. The scenario takes place at a music festival where a large number of bands appear on several stages, but also several stand-up comedians perform, people can go to the movies, and many other activities take place. This three-day festival is organized every year, attracting around the 50,000 visitors, and takes place on a large event area. Below, the activities of John on a festival day are described to show how users can profit from a nearcast framework at such festival.

John is visiting the festival every year with a group of friends. This year, they all have installed the mobile festival application that enables them to use all kinds of services. After registering at the festival’s website, John was able to download the festival application and install it on his mobile phone. This mobile nearcast application is required to be able to consume the nearcast services on the festival.

A week before the festival, John chooses his favorite artists and bands and adds them to his own schedule, which can be modified via the festival’s website or the nearcast application. There is also a utility to transfer or synchronize this schedule to the festival application on the phone. During the festival, John uses this schedule to check which performances he really wants to see and he receives reminders before each of these performances begins. John also creates a contact list and a community. Members of this community may post and view each other’s messages, uploaded photos, videos etc.

When John and his friends arrive at the festival, they start the nearcast application to see what is currently going on at the different stages. Each performance displays its popularity that is measured by the number of people that have it scheduled in their own schedule or the number of mobile phones discovered by the nearcast center(s) at the corresponding stage. One of the currently performing bands has a high popularity and John and his friends want to check out why. They look on the map of the festival area (using the nearcast application) to find out where they have to go and where they can pick up some drinks. They arrive at the stage where the band is appearing and enjoy the first performance. Using the nearcast application, John gives this performance a 4-star rating. After this warming-up, they walk around on the festival area and check out some other performances. At each stage, the nearcast application shows which artist or band is performing with links to its biography, discography and other related information.

Then, John receives an alert saying that one of his favorite bands starts performing in a quarter of an hour at stage Bravo. Using the administration interface, the festival
organizer has configured the system in such way that all visitors who have the band in their schedule will receive a message that contains an offer that gives discount on all the band’s merchandise, which is sold around the stage. John receives this message too.

Some of John’s friends want to go to another performance so they decide to split up. During the performance, John records a video of the band with his mobile phone and sends it to his friends using Bluetooth and the nearcast application. His friends answer that they are also enjoying their time.

When the performance is finished, John activates the nearcast application and checks the contact list to view where his friends are. Because they are on the other side of the festival area, he decides to send them a message to meet them halfway. They reply that they agree and want to meet in half an hour.

Then, the festival’s organization is informed that one of the bands has some delay. The organization decides to postpone the performance of this band with one hour and wants to inform all visitors with this information. They go to the nearcast administration application, make the changes in the schedule and choose the option to inform all nearcast users. While John is walking in their direction, he receives this message from the organization that says that one of the performances is delayed. The schedule in the nearcast application is updated automatically. John also browses the festival’s forum where visitors or other interested people may discuss different topics about the festival. This forum is accessible via the nearcast application but also from the festival’s website.

In the afternoon, John and his friends take a break for a while. They sit down in the lounge room, which they have found using the map in the nearcast application, and relax for a moment. While relaxing, they play several games of poker via the nearcast application.

The scenario above shows how users may profit from the nearcast framework. Although it describes a music festival, the nearcast framework is also suitable for deployment at other kinds of festivals, exhibitions or conferences.

### 6.2 Domain model

This section presents the domain model of the nearcast framework to give an understanding of the nearcast framework’s key concepts and vocabulary (Larman, 2004). The domain model for the nearcast architecture (Figure 9) shows that each user has a mobile device, e.g. a mobile phone or PDA, to be able to use the nearcast framework. In order to let users communicate with the nearcast center, this device needs to have the nearcast application installed, which is distributed by the nearcast center. The nearcast application communicates with the nearcast center that provides several services. A service can use multiple pieces of content and is aware of its context, including the device’s context, the user’s context and possibly the content’s context. Context is provided by the mobile device, nearcast application or nearcast center. Content is provided by users and content providers. Content providers also configure the nearcast services. Developers develop the nearcast centers and the nearcast applications. Service providers configure and install the nearcast centers. The model gives an overview of what functionality the nearcast framework should offer.
6.3 Use cases

Based on the scenario in section 6.1, the functional requirements for the nearcast framework are elicited. The functional requirements are captured in use cases. These use cases are described in a rather general way. Use cases describe how to achieve a business goal or task and they show interactions between the external actors and the nearcast framework.

Several stakeholders for the nearcast framework architecture have been identified. The main stakeholders are:

- **End-users**, who are the main users of the system: they have an application installed on their mobile phone that enables them to interact with the nearcast framework;
- **Content providers** want to create interaction with the end-users by providing content and services;
- The **Service provider** will often be the organization of an event and is responsible for determining what the application should look like and for finding content providers that want to provide content and services via the nearcast framework;
- **Developers** are responsible for the implementation and maintenance of the nearcast framework.

In the scenario described in section 6.1, the visitors of the festival are the end-users. The festival’s organization is the service provider, but the organization can also be one of the content providers and possibly the system administration. The organization should be able to install the nearcast framework themselves. This installation could also be left to the developers or another external party.

Figure 10 shows an overview of the architectural significant use cases. In the next sections, for each of these stakeholders, these use cases are elaborated.
6.3.1 End-users

In this section, the end-users’ use cases are discussed. Some of them have been made very abstract to keep the number of use cases limited. The ‘send message’-scenario, which allows users to create and send messages to other users, can be abstracted to the use case ‘upload content’, because sending a message is actually uploading a specific type of content that contains the message. Each use-case is numbered as a functional requirement (FR).

### FR1: Download application

<table>
<thead>
<tr>
<th>Main success scenario</th>
<th>Alternative scenario(s)</th>
</tr>
</thead>
</table>
| Nearcast center pushes messages to nearby devices that are not registered yet. This message contains a link to the internet where the user can indicate to download application. When the nearcast framework is notified that the user wants to download the application, it will push the application to the user. | 1. The nearcast center is configured to push the nearcast application to all nearby mobile devices, except those who are registered already;  
2. The nearcast center is able to receive objects from nearby devices. When a device pushes an object (someone’s business card for example) to the nearcast center, the nearcast center pushes the nearcast application to this device;  
3. By sending an SMS to a specific phone number, users are also able to request the application. The nearcast framework subsequently sends a WAP push message that contains a download link to the application;  
4. Users are also able to download the application using a normal PC web browser and to put it on the phone themselves;  
5. Another way is the user downloading the nearcast application with the... |
mobile phone’s browser from the festival’s website directly to his phone. This is a summary of the currently possible methods. The design of the nearcast framework should take into account that other methods may become available in the future.

**Exceptions**
The nearcast system should not send messages to nearby devices that are not able to run the nearcast application, headsets for example.

**Rationale**
With the nearcast client application installed, users are able to interact with the nearcast framework in a more user-friendly and more interactive way than without application.

### FR2: Download content

| Main success scenario | 1. User selects a piece of content from the menu in the nearcast application;  
| 2. The nearcast application sends a request for this piece of content to the nearcast center;  
| 3. The nearcast center returns the requested piece of content;  
| 4. The nearcast application opens the downloaded piece of content. |

| Alternative scenario(s) | When the user does not have the nearcast application installed, he should be able to request a piece of content by sending an object (a note or business card for example) to the nearcast center manually. The nearcast center should be configured to return the requested content. |

| Exceptions | 1. The nearcast application cannot find a nearcast center. In this case, the nearcast client application is in ‘offline’-mode. When the user selects the content to download, he will receive a notification that the nearcast center is unavailable and the content will be downloaded as soon as the user is in range of a nearcast center again;  
| 2. The transfer of the content fails. This could happen when the user becomes out-of-range of the nearcast center or after some system error. The user receives in such situation a notification that the transfer has failed. |

| Rationale | The nearcast framework can be used to download content. This content may vary from text, business cards, calendar events, promotions, pictures, screensavers, animations, ring tones, audio, video and games to applications. |

### FR3: Upload content

| Main success scenario | 1. The user selects a piece of content to upload using the nearcast application;  
| 2. The application uploads the content to the nearcast center;  
| 3. The nearcast center stores the content.  
| The nearcast center can do several things with the uploaded content:  
| 1. Publish content on a website;  
| 2. Publish content on a large screen next to the nearcast center;  
| 3. Forward content to another user (to realize a messaging system); |

| Alternative scenario(s) | The upload scenario is like the download scenario also applicable for devices that do not have the nearcast application installed. Users can |
send pictures or videos to the nearcast framework using the native phone’s interface to send files to other devices by Bluetooth. The disadvantage of this method is that the user cannot send meta-information along with the content and hence cannot specify what the nearcast framework should do with the sent content.

Exceptions
Same as FR2.

Rationale
As well as downloading content, users are able to upload content to the nearcast platform. This content may consist of the same types of content as in the download scenario.

<table>
<thead>
<tr>
<th>FR4: Content push</th>
</tr>
</thead>
</table>
| **Main success scenario** | 1. The nearcast center discovers a device that meets the requirements for a specific piece of content (configured by a content provider);  
2. The nearcast center pushes the content to the nearcast application;  
3. The nearcast application notifies the user of a new message and presents a shortcut to open the received content. |
| **Alternative scenario(s)** | In case a device does not have the nearcast application installed, the mobile phone will ask the user to accept the incoming message. After confirmation, the content is either stored on the device or opened automatically depending on the type of device. |
| **Exceptions** | If the device is out-of-range again, the content push will fail. |
| **Rationale** | Using content push, the nearcast system is able to push news updates or deliver user messages to the nearcast clients. |

<table>
<thead>
<tr>
<th>FR5: Content pull</th>
</tr>
</thead>
</table>
| **Main success scenario** | 1. The nearcast system has a connection with a nearcast client device and needs more information about this device;  
2. The nearcast system sends a request for a piece of content;  
3. The nearcast application returns the requested information. |
| **Alternative scenario(s)** | When the nearcast system does not have a connection with the client device, the system is still able to retrieve information about the device via the Bluetooth service discovery protocol. |
| **Exceptions** | Same as FR4. |
| **Rationale** | Using content pull, the nearcast system is able to retrieve context information about users or their devices (see section 6.4 for different types of context information). |

<table>
<thead>
<tr>
<th>FR6: Subscribe service</th>
</tr>
</thead>
</table>
| **Main success scenario** | 1. User chooses in nearcast application the option to subscribe to updates of a specific service;  
2. The nearcast application sends the subscription information to the nearcast center. |
| **Alternative scenario(s)** | The user is also able to subscribe to a service via the internet:  
1. on his mobile phone;  
2. on a PC |
| **Exceptions** | When the subscription fails, the user should receive a notification. |
| **Rationale** | By subscribing to services, users are able to indicate their interest in... |
specific information. These subscriptions can be used again as context information.

6.3.2 Content providers
This section elaborates the content providers’ use cases.

**FR7: Configure nearcast services**

<table>
<thead>
<tr>
<th>Main success scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Content provider starts administration application;</td>
</tr>
<tr>
<td>2. Content provider adds or selects an existing nearcast services;</td>
</tr>
<tr>
<td>3. Content provider configures the behavior of this service.</td>
</tr>
</tbody>
</table>

**Alternatives**
N/A

**Exceptions**
When the administration application is unavailable, the user should be notified.

**Rationale**
Content providers should be able to configure nearcast services. A basic nearcast service could be the distribution of a specific piece of content. Possible options to be configured are how many times content has to be distributed and to what kind of devices (filtering on client device capabilities).

**FR8: Upload, create or modify content**

<table>
<thead>
<tr>
<th>Main success scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Content provider starts administration application;</td>
</tr>
<tr>
<td>2. Content provider chooses relevant service;</td>
</tr>
<tr>
<td>3. Content provider uploads or creates content.</td>
</tr>
</tbody>
</table>

**Alternatives**
N/A

**Exceptions**
Same as FR7.

**Rationale**
Content providers may want to upload their own content (like multimedia files) or create new content (news messages) to be used in the nearcast services. Specifically for the music festival scenario, content providers should be able to update stage information, the facilities, the schedule and all performing artists.

**FR9: View statistics**

<table>
<thead>
<tr>
<th>Main success scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Content provider starts administration application;</td>
</tr>
<tr>
<td>2. Content provider chooses which statistics he wants to see.</td>
</tr>
</tbody>
</table>

**Alternative scenario(s)**
The statistics could include information about:
- Which devices have been seen around the nearcast centers;
- Which services these devices have been using;
- How many times content has been downloaded, uploaded or pushed;
- What devices have accepted or declined messages.

**Exceptions**
Same as FR7.

**Rationale**
By providing statistics, content providers are able to track how many users their service have, such that they can add or remove services for optimal benefit.

6.3.3 Service provider
This section elaborates the service providers’ use cases.

**FR10: Deploy and configure nearcast system**
### Main success scenario

1. Service provider places all nearcast centers in the environment and connects them to the internet;
2. Service provider starts administration application;
3. Service provider is able to view the connected nearcast centers;
4. Service provider is able to modify associated information of these nearcast centers;

### Alternative scenario(s)

N/A

### Exceptions

Same as FR7.

### Rationale

Service providers should be able to install, deploy and configure the nearcast system themselves.

---

### FR11: View and modify privileges

<table>
<thead>
<tr>
<th>Main success scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Service provider starts administration application;</td>
</tr>
<tr>
<td>2. Service provider selects a content provider from the list of content providers;</td>
</tr>
<tr>
<td>3. Service provider modifies privileges of the selected content providers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alternative scenario(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same as FR7.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>The content providers’ privileges concern the number of services they may create and on which nearcast centers these services may run.</td>
</tr>
</tbody>
</table>

---

### 6.3.4 Developers

This section elaborates the developers’ use cases.

---

### FR12: Create and deploy mobile nearcast application

<table>
<thead>
<tr>
<th>Main success scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Developer configures which components should be included in the nearcast application;</td>
</tr>
<tr>
<td>2. Developer configures the variation points of these components;</td>
</tr>
<tr>
<td>3. Developer configures look and feel (logos, colors and icons) of the nearcast application;</td>
</tr>
<tr>
<td>4. Developer creates different versions of the applications intended for different types of mobile phones;</td>
</tr>
<tr>
<td>5. Developer uploads application to nearcast system that is responsible for further distribution of the this application.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alternative scenario(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Different content providers require different functionality for their application</td>
</tr>
<tr>
<td>• The developer should be able to create a branded application for each content provider, such that users associate the application with the content provider.</td>
</tr>
<tr>
<td>• Different mobile phones have different operating systems, support different programming languages and have different capabilities. Therefore, an application is required for each device or group of</td>
</tr>
</tbody>
</table>
FR13: Create new service

<table>
<thead>
<tr>
<th>Main success scenario</th>
<th>1. Developer modifies the code to implement the new service;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Developer deploys the new software components to the different nearcast components.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alternative scenario(s)</th>
<th>N/A</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>N/A</th>
</tr>
</thead>
</table>

| Rationale | The nearcast framework can be used in many different scenarios, each requiring its specific services. Developers should be able to easily extend the capabilities of the nearcast framework by developing new services. |

6.4 Context-aware requirements

Section 4.3 introduced a categorization of context into primary and secondary context information. Primary context information consists of the identity, location, activity and time of an entity. These types of context information can be used on their own, but primarily as indices to find secondary context.

Several entities can be identified in the nearcast framework: users, devices, stages and content. This section analyzes for each entity what kinds of secondary context information the nearcast framework should support and how the nearcast framework can use this information in order to determine the benefits of the context-aware features of the nearcast framework.

6.4.1 Users

Table 7 shows what secondary context information the nearcast framework should gather about its users. For each piece of context information, this table also suggests how the nearcast system can use this information.

<table>
<thead>
<tr>
<th>Context information</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s name</td>
<td>Used for communication between users</td>
</tr>
<tr>
<td>Contacts</td>
<td>Enable users to send messages to existing contacts</td>
</tr>
<tr>
<td>Calendar</td>
<td>Enable users to schedule performances to be reminded at the festival when this performance almost starts</td>
</tr>
<tr>
<td>Music preference</td>
<td>Enables nearcast system to send user-tailored information. For example: when the nearcast system knows the user’s music preference or favorite bands, it can send information related to this music preference or these favorite bands</td>
</tr>
<tr>
<td>Mood/Status</td>
<td>Used for communication between users. If a user wants to contact another user and knows the other’s mood or status, he can decide how to contact this user or to contact this user at a later moment.</td>
</tr>
<tr>
<td>Location history</td>
<td>Determine when and where a user has been; this information can be used to send user-tailored content. For example: if a user is for the first time at a specific location, the system may send a different message than when the user has been on that spot earlier.</td>
</tr>
<tr>
<td>Transaction history</td>
<td>Can be used to send user-tailored content. For example: when a user</td>
</tr>
</tbody>
</table>
already received a token for a discount, the system can decide not to send this token again.

| Social situation | Can be used to send group-tailored content. For example: when the system has recognized a group of users as a group, because they have been seen several times at the same spot, it can send specific discounts for groups of people. |

6.4.2 Devices

The primary context information of a device consists of the Bluetooth address, its location and time. Table 8 presents an overview of secondary context information of the users’ devices in the nearcast framework.

Table 8 - Overview of context information about devices in the nearcast framework and how this information can be used.

<table>
<thead>
<tr>
<th>Context information</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer and model number</td>
<td>This information can be used to determine which version of the application the nearcast framework has to send to the corresponding device.</td>
</tr>
<tr>
<td>Operating system, installed API’s, camera support</td>
<td>This information can be used to determine what functionality the nearcast application can use.</td>
</tr>
<tr>
<td>Screen size, color capabilities and screen resolution, file type support</td>
<td>Can be used to determine in what format content should be distributed to the corresponding device.</td>
</tr>
<tr>
<td>Signal strength</td>
<td>Can be used to warn the user that he almost becomes out-of-range of the nearcast center.</td>
</tr>
</tbody>
</table>

6.4.3 Nearcast center

Primary context information of the nearcast center includes the name and location of the nearcast center and related information about that location or what activity is taking place. Table 9 presents an overview of secondary nearcast center context information.

Table 9 - Overview of secondary nearcast center context information

<table>
<thead>
<tr>
<th>Context information</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearby devices</td>
<td>The nearcast center can report or connect to nearby devices.</td>
</tr>
<tr>
<td>Transaction history</td>
<td>Can be used to determine what content has been distributed and what content should going to be distributed.</td>
</tr>
<tr>
<td>Other nearcast centers</td>
<td>Can be used to determine at which locations a user has been before. This information can be used to determine whether to send a specific piece of content or not.</td>
</tr>
<tr>
<td>Schedule information</td>
<td>The schedule information contains the schedule for performances. These can be distributed to the users.</td>
</tr>
</tbody>
</table>

6.4.4 Content

Instead of using the transaction history of a nearcast center (from section 6.4.3), the system may also use the transaction history of the content itself. The primary context information of content comprises the name of content and the time and location where it is created. Table 10 presents an overview of secondary context information of content that could be used within the nearcast framework.
Table 10 - Overview of secondary context information of nearcast content

<table>
<thead>
<tr>
<th>Context information</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction history</td>
<td>The transaction history of a device includes how many times, at which locations and to which users the system has distributed the corresponding piece of content. With this information, the nearcast framework can create a popularity list and present this list to users or send once in a hundred times a piece of content that contains a prize or discount.</td>
</tr>
</tbody>
</table>

Furthermore, context information may overlap different events. For example, if a festival’s visitor goes to another festival several weeks later, the nearcast framework should be aware of the context information of the previous festival.

6.5 Quality requirements

This section presents the driving qualities for the nearcast framework: flexibility, reliability and performance. Each quality includes multiple quality attribute scenarios that first describe a situation that can occur within the system and second, specify the required response of the system (Bass et al., 2003). By testing these scenarios against the proof-of-concept implementation, the software architecture can be validated. This validation is subject in chapter 8.

6.5.1 Flexibility

The flexibility requirements of the nearcast framework consist of three aspects. First, it should be easy to build new services. Second, other systems should be able to integrate the nearcast framework easily. Third, the short-range communication technology should be easily replaced with another technology. This section further elaborates these requirements.

QR1: Flexibility in development of new services

*The developers want to add a service to a nearcast system. The changes will be made to the code at design time and will affect no more than six modules.*

The nearcast framework can be deployed for different events or festivals, each with its own characteristics. These different characteristics imply that every event may provide services that are specific for that event. For the music festival in the example scenario, the schedule service can be a very useful service. However, if the nearcast framework is used in a different scenario, this service could be worthless, while other services may exist that are of unique value for that scenario. In addition, new services may be devised as the framework is tested in practice. It is therefore impossible to create a nearcast framework that supports all services one could think of. The goal of the nearcast framework is thus to create an environment in which developers are flexible and find it easy to develop new services. Therefore, the number of affected modules should be as low as possible.

QR2: Integration with external systems

*An external system’s developer wants to integrate the nearcast system in its own system. The service provider adds an account for this system, after which the external system’s developer is able to access the nearcast framework according to the specified interface. The nearcast system does not need to be modified or restarted.*
Another aspect of flexibility is that other systems should be able to integrate the nearcast framework easily. An external system can use the nearcast framework as an extra communication channel with their clients. Users can upload content to the nearcast framework, which in its turn forwards this content to the external system. On the other hand, external systems can use the nearcast framework to distribute content. The requirement is that an external system is able to integrate the nearcast framework, just by creating an interface with the nearcast framework, without changing any code or restarting the system. In such situation, the nearcast center is not affected when an external systems need to be integrated during an event.

QR 3: Communication technology independence
The developer wants to replace Bluetooth with another technology. The changes to the code will be made at design time and will affect only the communication modules of the nearcast client and nearcast center.

The nearcast center and the nearcast client application communicate with each other using a short-range communication technology. Potential technologies are Bluetooth, WiFi and RFID. There is no one best solution, because the nearcast framework can be used in different situations. It is also possible that within several years emerging technologies become better alternatives. Therefore, the nearcast framework should be prepared for any short-range communication technology.

6.5.2 Reliability
Two scenarios determine the reliability of the nearcast framework:

QR4: Connection robustness
Users initiate 360 requests per minute stochastically on the nearcast center that is operating normally; 99% of these requests arrive at the nearcast center correctly.

The robustness of the connection between the nearcast client and nearcast center is one of the two factors that mainly determine the reliability of the nearcast framework. This quality scenario indicates how well this connection can be used. One of the side issues of this scenario is that the connection should be re-established automatically, if the user is within the range of a nearcast center.

QR5: Discovery reliability
The nearcast center continuously scans its environment for nearby devices. The nearcast center should report 99% of the devices that are actually within its range, under all circumstances.

Another factor is the discovery reliability. Cheverst et al. (2005) have encountered several problems with the Bluetooth discovery. According to their research, in an environment with many Bluetooth devices, not all devices are reported. For example, in one case, only five or six of the ten devices were reported.

6.5.3 Performance
The nearcast framework’s performance is determined by the connecting time, delivery time, transfer rate and capacity of the nearcast framework.

QR6: Connecting time
A nearcast client enters the range of a nearcast center. The nearcast center is already connected to 30 devices. The new device is connected within 5 seconds.

It is important to have a low connecting time, because visitors can only use the nearcast application when they are connected to the nearcast framework. This scenario also takes into account a high capacity of the nearcast framework. Aalto et al. (2004) have performed extensive measurements for the discovery delay of the B-MAD system, which is the time between the user entering the range of the system and registration. They concluded that this discovery delay could be decreased by prolonging the inquiry process and instead of waiting for the whole inquiry process to be completed, returning the addresses of the found devices immediately when they are found.

QR7: Latency
A user sends a small message, less than one kilobyte, to another user, who is located at another nearcast center. The nearcast systems are both connected to 30 devices. The message is delivered within 5 seconds.

The network latency is measured by how long it takes for the system to handle and deliver messages. This is an important measure, because it indicates how reactive the system is. For example, when a user requests some information from the nearcast center, this measure can be used to determine how soon the request is delivered again at the user.

QR8: Throughput
Thirty users are simultaneously downloading or uploading content from the nearcast center. The throughput for each connection is at least 25.7 KB per second.

It is important that the throughput between the nearcast client and nearcast center is high enough and that the full bandwidth of Bluetooth is used. The Bluetooth version 2.0 specification states a gross air bit rate of 3 Mbps, i.e. 375 KB per second. Bray [16] shows that the different layers of the Bluetooth stack may decrease the gross air bit rate with 52 percent, resulting in a gross bit rate of 180 KB per second. Using seven connections on one Bluetooth adapter, this results in a bit rate of 25.7 KB per second.

QR9: Scalability
The content provider wants the system to be able to handle 50,000 visitors simultaneously. The nearcast system is extended in such way that it has this capacity, with no decreased qualities.

The festival described in section 6.1, attracts around the 50,000 visitors a year. The nearcast framework as a whole should have the capacity that all these visitors can use the nearcast framework simultaneously. This capacity can be achieved by using multiple nearcast centers. Two of the compared systems have provided numbers for their capacity. The Bluecasting system is able to handle 154 simultaneous connections but provides only one-way communication, while the B-MAD system only supported one connection at the time.
6.6 Conclusions

This chapter has presented the requirements for the nearcast framework. The scenario, domain model and use cases should provide insight into the functionality of the nearcast framework.

The context-aware requirements discuss what context information should be used and how the nearcast framework should use this information. The goal of this analysis is to discover the benefits of context-aware services in the nearcast framework. Some examples of the context-aware services identified in section 6.4 are:

- The nearcast center knows the context of a discovered device and is thus able to decide which version of the nearcast application the user has to install.
- The nearcast center knows the context of the discovered devices and is thus able to decide whether to connect to each of these devices or not.
- The nearcast application knows the user’s context and is thus able to deliver services when the user finds is in a certain context.

Consequently, context-aware nearcasting enables content-providers to deliver unique and tailored content to the user, content that is adapted to the wishes and needs of the user. The more context information is available, the better the system knows who is around, what he or she likes and dislikes, where the person comes from and where he is now, the better the system can adapt its behavior. These are the benefits of the context-aware services of the nearcast framework.

The driving quality requirements for the nearcast framework are flexibility, reliability and performance.

- Flexibility means that new services can be created easily, other systems should be able to integrate the nearcast framework, and the communication technology between the user and nearcast center should be easily replaced.
- Reliability comprises of the requirement to be able to have consistent and stable connections between mobile devices and nearcast centers and that the discovery process detects almost all devices in range.
- Performance is measured by the connecting time, network latency, throughput and the scalability of the nearcast system.
7 Software architecture

Software architecture is concerned with the top-level decomposition of a system into its main components, the relations and communication between these components and the interfaces to the external world (Bosch, 2000). Software architecture comprises the earliest design decisions, which have a lot of impact on the system’s development, its deployment and its maintenance life. This chapter describes the software architecture for the nearcast framework using the architectural view model.

Kruchten (1999) introduced the 4+1 view model to describe and document the software architecture of a system. The 4+1 model consists of four plus one views: the logical-, process-, physical- and development view each describe specific aspects of the system plus the fifth view that describes the architectural significant use cases. All views together form the architecture.

The concept of architectural views can be illustrated using the analogy with a bird wing (Clements et al. 2003). There is not one view or drawing of a bird wing. Instead, one can show only one aspect of the wing, for example the feathers, the skeleton, circulation or muscles. This also applies for software architecture. A system cannot be described with one view or one diagram, but different views are needed that focus on different aspects of a system.

During the years, Kruchten’s four views have been expanded with a whole range of other views. Because there are so many views, the software architect has to choose which ones are relevant and most important by analyzing which aspects of these views are of importance for the involved stakeholders. The selected views will convey the system’s software architecture.

The stakeholders of the nearcast framework include current and future architects, the project manager and members of the development team. One of the quality scenarios concerns the integration with external systems. Therefore, designers of external systems are included in the stakeholders list. Other quality requirements are flexibility, reliability and performance. To validate the architecture, the analysts of these qualities need specific architectural documentation and are therefore included in the stakeholders list.

After the list of stakeholders is assessed, an overview is generated that summarizes the documentation needs of these different stakeholders. Shows part of this overview: only the selected views are included.

- Architects or developers are interested in all views in order to understand the complete architecture.
- A project manager will be interested in the layered view to obtain a general idea of the system. The decomposition view may help the project manager to assign the work to different developers. Further, he may be interested in the deployment view from which he can derive the hardware costs. Finally, the project manager may be interested in the interaction with other systems addressed by the client-server view, which may suggest an organization-to-organization interface.
- A designer of an external system is primarily interested in the interfaces that he can use to control the nearcast framework. These interfaces are specified in the
client-server view. Further, he might be interested in the global workings of the system that are addressed in the layered view.

- Analysts validate the architecture against specific quality attributes, such as flexibility, reliability and performance. A performance analyst will not be interested in the same views as a flexibility analyst. The layered decomposition view can be used to address the flexibility of the system. Both deployment and client-server view address various aspects of performance and reliability, while the context view also plays an important role for performance.

Table 11 - Architectural views and stakeholders

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Layered</th>
<th>Decomposition</th>
<th>Deployment</th>
<th>Client-server</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architect</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>Project manager</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>Developer</td>
<td>d</td>
<td>d</td>
<td>s</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>Designer external system</td>
<td>o</td>
<td></td>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Analyst for flexibility</td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyst for reliability</td>
<td>d</td>
<td></td>
<td>d</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Analyst for performance</td>
<td>d</td>
<td></td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

This table shows the nearcast framework stakeholders and which architectural views they find useful and to what extent: d = detailed information, s = some information, o = overview information. The architectural views are selected from (Clements et al., 2003).

This chapter continues to describe the selected views that convey the software architecture for the nearcast framework.

- The **Layered decomposition view** gives an overview of how functionality is assigned to the different components in the nearcast framework and how the system can be divided into layers (section 7.1).
- The **Deployment view** shows what hardware components the system contains and how software elements are mapped onto these components (section 7.2).
- The **Client-server view** shows how communication between clients and servers takes place and specifies the interfaces between the different components (section 7.3).
- The **Context view** shows how the nearcast framework handles context information (section 7.4).

### 7.1 Layered decomposition view

This view is a combination of the composition- and the layered view. The decomposition view shows how the nearcast framework is decomposed into different components, while the layered view shows of what layers the nearcast framework consists. These views are combined into one view to show the relation between the layers and the different components.
7.1.1 Primary representation

![Layered decomposition view of the nearcast client, center, and server](image)

Figure 11 - The layered decomposition view shows the three main components, the nearcast client, nearcast center, and the nearcast server, and the modules of which they consist. In addition, three layers are identified: the presentation, application and communication layer. The lines between modules represent that there is an interface between these modules.

7.1.2 Elements and properties

Figure 11 shows that the nearcast framework is decomposed into three different components and three different layers. The layers in each component contain one or modules again. Table 12 describes the components, Table 13 describes the layers and Table 14 describes the modules in the nearcast framework.

The external applications are not included in the layered decomposition view, because they are not part of the nearcast framework. However, the integration of these external applications and systems is an important requirement for the nearcast framework and will be discussed in the client-server view in section 7.3.

Table 12 - Description of the components in the nearcast framework

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearcast client</td>
<td>Responsible for presenting the user interface to the end-user, gathering context information about the user and the device, and communication between the nearcast client and the nearcast framework.</td>
</tr>
<tr>
<td>Nearcast center</td>
<td>Responsible for handling requests from the nearcast client and nearcast server, generating a user interface on the spot, and gathering context information about the nearcast center.</td>
</tr>
<tr>
<td>Nearcast server</td>
<td>Responsible for handling requests from other nearcast components or external systems, generating the administration interface for content and service providers, and storing context information.</td>
</tr>
</tbody>
</table>
### Table 13 - Description of the layers in the nearcast framework

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>Contains the user interface modules that handle the presentation of the application and processing of the user input. Because of the layered model, other presentation interfaces can be easily added.</td>
</tr>
<tr>
<td>Application</td>
<td>Contains the functionality to determine what the component should do. This includes the management of context information. For this layer also applies that changes do not need to affect other layers. For example, when another database is going to be used, this has no effects on the presentation and communication layers.</td>
</tr>
<tr>
<td>Communication</td>
<td>This layer is responsible for communication between the different components. The Broker pattern is applicable here, as the communication functionality of the nearcast system is separated from its application functionality (Avgeriou et al., 2006). The communication layer hides all communication between the different components. Whether the nearcast client communicates with the nearcast center by Bluetooth or WiFi does not matter to the application layer. Furthermore, the dispatcher contains a marshalling object that transforms the procedure call into a string that can be sent over the transmission medium (see section 7.3)</td>
</tr>
</tbody>
</table>

### Table 14 - Description of modules in the different components of the nearcast framework

<table>
<thead>
<tr>
<th>Component</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearcast client</td>
<td>User Interface(UI)</td>
<td>Handles the presentation of the application on the mobile device and passes user commands to the dispatcher.</td>
</tr>
<tr>
<td></td>
<td>Dispatcher</td>
<td>Responsible for controlling this component. It receives user commands from the user interface, receives or accesses context information from the context module, stores or retrieves information from the database module, and communicates with other components using the interface modules in the communication layer.</td>
</tr>
<tr>
<td></td>
<td>Context</td>
<td>Responsible for gathering, aggregating, processing and distributing context information. Contains sensors, aggregators and computational objects (see section 7.4)</td>
</tr>
<tr>
<td></td>
<td>Database</td>
<td>Responsible for storing configuration variables, context information and other data that should be persistent</td>
</tr>
<tr>
<td></td>
<td>Center Interface</td>
<td>Responsible for communication with the nearcast center component. Contains specific code for the Bluetooth communication, such that when this technology is replaced the application layer does not need to be changed.</td>
</tr>
<tr>
<td>Nearcast center</td>
<td>User Interface(UI)</td>
<td>Provides a user interface for end-users or content providers such that users can access the nearcast framework locally.</td>
</tr>
<tr>
<td></td>
<td>Dispatcher</td>
<td>Same as dispatcher module in nearcast client.</td>
</tr>
<tr>
<td></td>
<td>Context</td>
<td>Same as context module in nearcast client</td>
</tr>
<tr>
<td></td>
<td>Database</td>
<td>Same as database module in nearcast client</td>
</tr>
<tr>
<td></td>
<td>Client Interface</td>
<td>Responsible for communication with the nearcast clients. It contains specific code for the Bluetooth communication and is able to handle multiple clients simultaneously.</td>
</tr>
</tbody>
</table>
Server Interface
Responsible for communication with the nearcast server.
This is in fact the client in the client-server interaction
between the nearcast center and nearcast server.

Nearcast server
User Interface
(UI)
The user interface on the nearcast server is responsible for
generating the administration interface for the content and
service providers.

Dispatcher
Same as dispatcher module in nearcast client.

Context
Same as context module in nearcast client.

Database
Same as database module in nearcast client.

Center Interface
Responsible for communication with the nearcast centers.
This is in fact the server in the client-server interaction
between the nearcast center and nearcast server. The
module should be able to handle all nearcast centers’
connections.

External Interface
Responsible for communication with external applications.
This module could expose an interface via different
protocols, like sockets or HTTP, to achieve maximum
interoperability with external applications.

The interfaces between the components that define the communication between the
nearcast client, nearcast center and nearcast server are discussed in the client-server
view (section 7.3).

7.1.3 Relations
Table 15 presents the relations between the dispatcher module and the other modules of
the nearcast client component. The dispatcher is the central module that communicates
with all other modules in the component.

Table 15 - Interfaces between the dispatcher module and the other modules in the nearcast client component.
The dir-column shows whether the dispatcher calls the other module or vice versa.

<table>
<thead>
<tr>
<th>Module</th>
<th>Dir.</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface</td>
<td>►</td>
<td>notify()</td>
<td>Dispatcher notifies UI to update its view.</td>
</tr>
<tr>
<td></td>
<td>◀</td>
<td>handle_command()</td>
<td>UI passes a user command to the Dispatcher.</td>
</tr>
<tr>
<td>Context</td>
<td>►</td>
<td>get_context()</td>
<td>Get context information</td>
</tr>
<tr>
<td></td>
<td>◀</td>
<td>context_update()</td>
<td>Notify dispatcher that there is a change in the user’s context.</td>
</tr>
<tr>
<td>Dispatcher</td>
<td>►</td>
<td>_store_data()</td>
<td>Store information in database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_data()</td>
<td>Get information from database</td>
</tr>
<tr>
<td>Protocol</td>
<td>►</td>
<td>create_message()</td>
<td>Convert message to be ready to be sent to other components (i.e. create XML message)</td>
</tr>
<tr>
<td>Center Interface</td>
<td></td>
<td>send_message()</td>
<td>Send message to nearcast center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is_connected()</td>
<td>Determine status of connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cancel_download()</td>
<td>Cancel download operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connect()</td>
<td>Connect to nearcast center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>close()</td>
<td>Close connection with nearcast center</td>
</tr>
<tr>
<td></td>
<td>◀</td>
<td>handle_message()</td>
<td>Pass received message to dispatcher</td>
</tr>
<tr>
<td></td>
<td></td>
<td>handle_event()</td>
<td>Pass an event to the dispatcher</td>
</tr>
</tbody>
</table>
The relations between the modules in the nearcast center and nearcast server are not included in this thesis, because these are almost similar to the relations for the nearcast client.

### 7.1.4 Architectural design decisions

**AD1: Layered design**

The nearcast framework is designed following a layered architectural style, which decomposes a system into a set of horizontal layers. Each layer provides an interface to a higher-level layer for using the abstraction it represents. A pure layered model has good properties for modifiability in the sense that layers can be modified or replaced without changing their adjacent layers (Bosch, 2000).

### 7.2 Deployment view

The deployment view shows the mapping of software elements onto hardware elements and specifies what relationships between the physical elements exist and what properties these elements have. This view can be used to analyze performance, reliability and security of a system.

#### 7.2.1 Primary representation

![Diagram of deployment view](image)

Figure 12 - Deployment view of the nearcast framework. Note that the framework can consist of multiple nearcast centers and that the nearcast center can have multiple nearcast clients. Nearcast centers do not have a connection with each other.

#### 7.2.2 Hardware components

The hardware elements in the deployment view are mobile devices, nearcast centers and the nearcast server. The specific hardware properties for each of these hardware components are described below.

**Mobile devices**

The first category of hardware elements are the mobile devices, in particular mobile phones. One requirement is that the device should be equipped with Bluetooth in order to be able to communicate with a nearcast center. There are no other particular requirements for these devices, although it is recommended that the device has a (large) color screen to provide good user experience and an on-board camera to create content that can be shared within the nearcast framework.

**Nearcast center**

The nearcast center can be a regular PC or a mobile device. The particular hardware properties are that the nearcast center is required to have a broadband internet connection for communication with the nearcast server and it should have installed a
number of Bluetooth adapters to enable the communication with the nearcast client applications. The number of Bluetooth adapters determines the capacity of the nearcast center.

Each nearcast center is configured to handle a number of clients. When the nearcast center should be able to handle thirty users simultaneously for example, the system needs to have installed at least five Bluetooth adapters and the software should address these Bluetooth adapters correctly.

Nearcast server
The nearcast server can be a regular web server. It is required to have a broadband internet connection to be able to handle the requests from all nearcast centers.

7.2.3 Relations
The mobile devices have two communication channels with the nearcast server. The primary communication channel is the Bluetooth connection with the nearcast center. The bandwidth of this connection depends on the Bluetooth version that can be used. Using Bluetooth version 1.2, a bandwidth of 723kb/s is available, while Bluetooth 2.0 enables a maximum bandwidth of 3Mb/s. These numbers are theoretical maxima and will be smaller in practice. Furthermore, when multiple connections are placed on one Bluetooth adapter, the bandwidth is shared.

The nearcast center communicates with the nearcast server via the internet or a local intranet. This nearcast server has a fixed address. The mobile devices should also be able to communicate with the nearcast server directly, via GPRS, UMTS or any other mobile network technology, to support downloading the nearcast application via the internet. The bandwidth of the communication link with the nearcast center is typically higher and cheaper than the one with the nearcast server.

Both nearcast centers and nearcast server have high bandwidth links (10Mbit/s or better) with each other. The short-range communication link between the nearcast center and mobile device is also a high bandwidth link, but the link between the nearcast device and the nearcast server is typically much more limited and will be used only when there is no link possible with the nearcast center.

7.2.4 Mapping of software components onto hardware components
The deployment view maps the software components from section 7.1 onto the hardware components enumerated in the section 7.2.2. There is almost a clear one-to-one mapping of these software components onto the hardware elements. The nearcast client component is allocated at the nearcast client device, the nearcast center component is mapped onto each nearcast center and the nearcast server component is allocated at the nearcast server.

7.2.5 Relations with previous views
The deployment view relates to the layered decomposition view in the way that is shows on which hardware components the software components are mapped and the physical connections between these components. Thus, the nearcast client, nearcast center and nearcast server components correspond with each other in both these views.
7.2.6 Architectural design decisions

AD2: Use Bluetooth as nearcast technology
Bluetooth will be the communication technology between the nearcast client and nearcast center. The primary reason for this decision is that most mobile phones currently support this technology, while only some phones do support WiFi or RFID for example.

AD3: Use a server for communication between the nearcast centers
All communication between the nearcast centers flows through the central nearcast server. This facilitates the communication and the nearcast server keeps a central administration of the nearcast centers. Without a central server, communication would be more complex because all nearcast centers have to communicate with each other and keep the context information consistent.

AD4: Message-based communication
The nearcast clients and nearcast centers communicate using messages. Nearcast clients are mobile devices without any standard remote computing environment and thus rely on message-based communication. Furthermore, by using message-based communication, the communication layer can be separated from the application layer. Nearcast centers also use message-based communication with the nearcast server to increase the interoperability with other systems. In this way, the only requirement for external systems is that they are able to send and receive messages. The result is that external systems have the same abilities as nearcast centers. The messages will be based on remote procedure call patterns to achieve access transparency in the nearcast framework (Tanenbaum, 2002).

AD5: Use XML for communication
Communication between the components in the nearcast framework will be message-oriented. The message-oriented model does not specify in what language components communicate. Possible alternatives are JSON [13], XML [21] and YAML [2].

XML has grown to a widely used standard and many development tools are available such that XML will not cause many problems when integrating the nearcast system with other systems. One disadvantage of XML is the overhead in the size of messages that can be generated when many and long tags are used. JSON and YAML suffer less from such overhead, because they denote structure by using special characters or white-space indentation respectively. However, much less parsers are available for these languages.

The differences between the different languages are small. Since the interoperability for XML is better, the nearcast framework will use this language for communication between remote components.

7.3 Client-server view

This section discusses the client-server view of the nearcast framework. This view can be used to analyze performance.

7.3.1 Primary representation
The client-server view describes the interaction between clients and servers in the nearcast framework. Figure 13 shows the main components and connectors of this view. The nearcast server acts as a server for several clients. First, it has a connector with the
nearcast center. The nearcast center, in its turn, acts as a server for the nearcast clients. The second connector connects external applications with the nearcast server.

In a regular client-server view, clients only request services from the server. However, in the nearcast framework, servers are allowed to initiate actions on their clients. For example, when a nearcast client wants to send a message to another client that is located at another nearcast center, this message will flow from the client to the nearcast center to the nearcast server. The nearcast server determines at which nearcast center the target device is located and sends the message to this nearcast center, which in turn delivers the message at the nearcast client. Thus, instead of clients only invoking services from the server, the nearcast framework allows servers also to deliver services at the clients.

Figure 13 - Client-server view. The elements are components of the nearcast framework, while the lines between these components represent interaction between these components. An arrow from component A to B means that component A calls the component B.

The clients and servers in the nearcast framework communicate using asynchronous messaging (see AD7). Thus, a client sends a message to a server and continues with what it was doing, i.e. it does not block to wait for an answer.

The messages that are sent between the different components are based on the XML-RPC pattern (see AD4). Figure 14 shows an example message. This message could be sent by the nearcast center to the nearcast client to update a remote user’s status. The def-tag specifies the remote function that has to be called and the following nodes contain this function’s parameters. Section 7.3.2 shows the interfaces between the different components by defining the available functions.

```xml
<xml>
  <def>context_update</def>
  <type>location</type>
  <address>00:16:4e:da:fa:43</address>
  <time>2006-11-01 13:45:29</time>
  <location>Alpha</location>
</xml>
```

Figure 14 - Example of a nearcast message

These messages can be sent over different transport protocols. Between the nearcast client and nearcast center, messages are sent over Bluetooth’s RFCOMM protocol (see appendix B for more information about Bluetooth). When this communication technology is replaced with WiFi for example, only the communication layer has to be
changed, while the same interface can be used. Between the nearcast server and its clients, these messages may be transported via either sockets or HTTP.

Further, connections between the nearcast client and nearcast center are persistent and connections between the nearcast server and its clients should be persistent under specific circumstances (see AD13).

7.3.2 Element interfaces

Table 16 and Table 17 present the interfaces between the nearcast client and nearcast center. As elaborated already in the previous section, these components communicate using XML-RPC-based messages. Each procedure call from the tables below can be represented as a XML-RPC-based message. The available procedures between the nearcast center and nearcast client can be found in the tables below, while the procedures between the nearcast server and administration application or external applications are included in appendix C.

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>download_content(type, filename)</td>
<td>Download content with filename and type.</td>
</tr>
<tr>
<td>upload_content(type, filename, content)</td>
<td>Upload content with filename, type and content</td>
</tr>
<tr>
<td>context_subscribe(type, values)</td>
<td>Let client subscribe for context updates</td>
</tr>
<tr>
<td>context_unsubscribe(type, values)</td>
<td>Let client unsubscribe for context updates</td>
</tr>
<tr>
<td>send_message(to_addr, time, msg)</td>
<td>Let client send a message msg to device to_addr.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>context_update(type, name, value)</td>
<td>Send context update from center to client</td>
</tr>
<tr>
<td>return_content(filename, content)</td>
<td>Return content and filename</td>
</tr>
<tr>
<td>send_msg(addr, name, date, body, sent_flag)</td>
<td>Deliver a message at the nearcast client</td>
</tr>
</tbody>
</table>

7.3.3 Element behavior

This section discusses the behavior of some elements in the client-server view. First, the connection establishment between the nearcast client and nearcast center is discussed and then the asynchronous communication between these elements is discussed.

Connection establishment between nearcast client and nearcast center

Figure 15 shows the sequence diagram of the device and service discovery. The device discovery, service discovery and interface are three separate processes. The device discovery starts and calls the service discovery process as soon as a device is found (see AD9). In this example, there are three devices discovered (A, B, C). The service discovery process determines whether each one of these devices is running the nearcast application by looking up the Bluetooth service discovery record of the device. If the nearcast service is found, the interface is called to establish a connection with the device. In the example, device A and B are found to have the nearcast service and the client interface is called to connect to these devices. Device C does not have the nearcast application installed and no further action is taken on this device.

When the device discovery finds two devices successively, the service discovery should queue these devices to discovery their services, because the service discovery can only perform one discovery at the time.
In the second discovery, device A is found again. Since the application knows that this device is already connected to the nearcast center, it does not send a connect-command to the client interface. Device C is also found again. Although the nearcast center already performed a service discovery on this device, it may have installed the nearcast application in between. Therefore, the service discovery process has to perform a service discovery on this device again. This time, the nearcast service is found on device C and passed to the client interface.

**Figure 15 - UML Sequence diagram of device and service discovery.**

**Asynchronous communication**

All elements in the nearcast framework support asynchronous communication. Figure 16 presents an example, where two devices A and B, located at the different nearcast centers X and Y, send a message to each other simultaneously. Nearcast client A sends a message to nearcast center X, which tries to locate the recipient of the message, nearcast client B. If this client would be connected with nearcast center X, the message could be delivered directly, after which the sending client and server are notified.

**Figure 16 - UML Sequence diagram of sending a message**
However, in this case, the nearcast client is located at nearcast center Y. Therefore, the message is sent to the nearcast server, which in turn forwards to nearcast center Y. This nearcast center delivers the message.

To show the asynchronous communication, the sequence diagram also contains the flow of sending a message from nearcast client B to nearcast client A. The nearcast clients and nearcast centers do not wait for a delivery notification, but are able to receive other messages immediately.

7.3.4 Relations with previous views
The elements in this view, the client-server view, clearly correspond with the software components in the layered decomposition and deployment view. The nearcast client in this view is thus similar to the nearcast client in the layered decomposition view and the nearcast client in the deployment view.

7.3.5 Architectural design decisions
AD6: Nearcast center connects to nearcast clients
One aspect of the client-server view is how connections are established between the clients and server. Normally, the client knows the address of the server beforehand and is therefore able to connect to the server quite easily. The nearcast framework, however, has to deal with ad-hoc networks where clients do not know the server beforehand.

The Bluetooth technology incorporates a so-called device and service discovery, with which a Bluetooth enabled device is able to discover nearby devices and what services these devices provide (see appendix B for more information about Bluetooth). However, performing these discoveries requires a lot of the device’s power consumption. When users walk around on a festival and their mobile device is scanning the environment regularly for a nearcast center, the mobile device’s battery will be dead very quickly. Therefore, this section studies the alternative way: let the server search and connect to client devices.

Each nearcast center already scans the environment for nearby devices, which results are used as context information to determine where and when a device has been seen. However, this information can also be used to find devices that are running the nearcast application and to establish a connection with these devices. The application on the mobile device should insert a record in its service discovery record and the nearcast center should not only report the address and name of nearby devices, but also perform a service discovery. The main advantage of this method is that client devices do not need to waste power by continuously performing device and service discoveries, but only need to publish the nearcast service. The drawback of this method is that the nearcast center needs to perform additional service discoveries, which may result in higher connecting times.

AD7: Use Bluetooth Service Discovery to let devices find each other
Another aspect of the client-server view that is related to how these devices connect with each other is how the nearcast center determines to which devices it should connect. There are two possible solutions. First, the nearcast framework keeps a list of devices that have a nearcast client installed. When the nearcast center discovers nearby devices, it can hold these addresses against this administration. Second, the nearcast client application publishes a nearcast service in the device’s service discovery records,
which can be retrieved by the nearcast center. The last solution is the most flexible and simple one and will be therefore used in the nearcast framework.

However, this solution can use some optimizations. The nearcast center should keep record of the devices it has scanned for services. If the nearcast service is found on a device, this is recorded. The next time this device is discovered, the nearcast center immediately knows it is a nearcast-enabled device and does not need to discover services again. This saves a lot of time. On the other hand, if a device does not publish the nearcast service, the nearcast center cannot skip the device the next time, because it could have installed the nearcast application in the meantime.

Further, the nearcast center can use a device’s context information. For example, the nearcast center can ignore devices that cannot install a nearcast application, such as Bluetooth headsets.

**AD8: Asynchronous communication**
The communication between the servers and clients is asynchronously, i.e. the clients do not wait for an answer after sending a message to the server. Asynchronous communication is more complex than synchronous communication, but it increases performance. Processes do not have to wait for a certain transfer to be completed, but may continue with their next operations.

**AD9: Run Bluetooth discovery for 15.36 seconds**
The Bluetooth specification states that it takes at least 10.24 seconds to complete a device discovery. However, in an error-prone environment, this inquiry time may be prolonged to increase the probability of receiving all responses. On the other hand, to guarantee that devices are discovered within a specific period, the device discovery is not allowed to take too much time (because the nearcast center needs to perform a service discovery often enough). Therefore, the device discovery should take 15.36 seconds.

**AD10: Bluetooth discovery returns found devices one by one**
There are two ways of performing the Bluetooth device discovery. Either the application returns all devices at the end of the discovery or the discovery returns the device information as soon as the device is found. The last method’s implementation is somewhat more complex due to its asynchronous nature, but it results in an earlier registration of nearby devices, which in its turn decreases the connecting time.

**AD11: Separate device and service discovery**
Many applications include looking up names in their discovery process. This may result in a much longer discovery process, because devices may become unreachable when they get out-of-range for example. The discovery process in the nearcast framework may not take more time than specified, thus to prevent the lengthening of this process, the nearcast center should separate the device and service discovery and use different adapters for these processes.

**AD12: Different connections are addressed on different Bluetooth adapters**
If the nearcast center connects to multiple devices, these connections are allocated to different Bluetooth adapters as much as possible, to ensure optimal data rates. The more connections on one Bluetooth adapter, the less bandwidth is available for each connection.
AD13: Use persistent connections

Figure 13 shows three different connectors for the different components in the nearcast framework. There are two different strategies to maintain these connections. Either a new connection is established for each request or a persistent connection is created that stays alive as long as the two components may communicate with each other. These concepts can be compared with connection-oriented and connectionless services (Tanenbaum, 2000).

The connection between the nearcast client and nearcast center will be persistent, because creating a connection via Bluetooth for each request takes much time and will not always be successful. According to AD5, only the nearcast center is allowed to establish a connection with the nearcast client. When both elements are allowed to establish a connection, this may result in difficulties, because if two devices are discovering they will not find each other. Thus, when the nearcast client is trying to discover a nearcast center, while the nearcast center is discovering the nearcast client, they will not find each other. This idea could work, however, when the nearcast center has multiple Bluetooth adapters. Then, the software has to make sure that the listening socket is addressed on a different Bluetooth adapter as the discovering socket is.

Thus, devices will stay connected with a nearcast center as long as they are close enough to this nearcast center. The disadvantage of this approach is that the capacity of one nearcast center is decreased: not all devices will communicate constantly with the nearcast center, but they hold a position on a Bluetooth adapter. The advantage however, is the lower latency of message delivery.

The solution to let the nearcast center poll its clients for new messages is not an option, because this results in too much loss in performance. If the client polls for new messages each 30 seconds, the delivery time will also be 30 seconds in a worst-case scenario. With these numbers, the system cannot satisfy the quality requirement.

The connection between the nearcast center and nearcast server should be persistent if the nearcast server cannot address the nearcast center directly, e.g. if the nearcast center is behind a firewall such that the server cannot determine its local IP address. In this case, a persistent connection should be created: the nearcast center establishes a connection with the nearcast server at startup and maintains this connection as long as the nearcast center is active. Both nearcast center and nearcast server can use this connection to send and receive data.

7.4 Context view

This section presents the context-view that shows how the nearcast framework handles context information, which can be used to analyze part of the nearcast framework’s performance, because the allocation of the context components influences what communication should take place.

7.4.1 Primary representation

The context view is based on the COOL framework and consists of the following elements: sensors, connectors, aggregators, object repository, services. These elements are described in section 4.6.4. This section will show an example configuration of the context view to illustrate the handling of context.
Figure 17 shows the elements that are used for the contact list service, which allows users to view a list of their friends and their corresponding location, music preferences and social situation. The configuration includes three different sensors, a location sensor, a music preference sensor and a social situation sensor, which are responsible to measure the corresponding context information.

The location and social situation sensor are implemented in the nearcast center. The nearcast center scans its environment continuously for nearby devices. Using this information, the location of devices can be determined, but also their social situation, assuming that the nearby devices of a nearcast center represent the social situation of a device. The music preference sensor is implemented in the nearcast application and measures the users’ music preferences by checking which performances the user has scheduled.

The sensors pass their information to the corresponding connector, which is responsible for converting and integrating the sensor data into the nearcast framework. These connectors run at the same device as the sensor, thus, the location and social situation connectors run on the nearcast center while the music preference connector runs on the nearcast client. Other components, possibly on other devices, may request data from these connectors or subscribe to be updated when the context information changes. In this example, the location and music preferences connectors have a related component that runs on the nearcast server. The social situation context information has no corresponding connector, because this information is only stored at the nearcast center.

The contact list service runs on the mobile device. When the user activates this service, the nearcast client application uses the object repository to find the needed components. Then, the service requests all context information and subscribes for updates in any of the context information. The location and music preferences components subsequently subscribe to the corresponding connectors for updates. When the sensors measure new information and passes this information to the connectors, these connectors know that
they should notify their subscribed components, which notify the contact list service in their turn. This information flow thus follows the publish-subscribe pattern.

### 7.4.2 Allocation of context information

This section discusses what context information can be used in the music festival scenario and shows how this information is allocated at the different components in the nearcast framework. This information can be used to analyze part of the nearcast framework’s performance.

Nearcast services may use a wide variety of different types of context information. presents a list of context information types that can be used in the music festival scenario. The entities and attributes are already introduced in section 6.4. The attributes are basic context information types and have a direct source that determines its current value. These attributes belong to an entity, which can be a person, a place or an object. Entities and attributes can be represented by aggregators and connector from the COOL framework respectively. In the example in the previous section, the location attribute is represented by a location connector, as the user entity is represented by the user aggregator.

Table 18 - Overview of context information

<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
<th>Source</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Identity</td>
<td>Nearcast application</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Contacts</td>
<td>Nearcast application</td>
<td>Nearcast application</td>
</tr>
<tr>
<td></td>
<td>Calendar</td>
<td>Nearcast application</td>
<td>Nearcast application</td>
</tr>
<tr>
<td></td>
<td>Schedule</td>
<td>Nearcast application</td>
<td>Nearcast application</td>
</tr>
<tr>
<td></td>
<td>Music preference</td>
<td>Nearcast application</td>
<td>Nearcast application</td>
</tr>
<tr>
<td></td>
<td>Mood</td>
<td>Nearcast application</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Location</td>
<td>Nearcast center</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Transaction history</td>
<td>Nearcast center</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Social situation</td>
<td>Nearcast center</td>
<td>Nearcast center</td>
</tr>
<tr>
<td>Stage</td>
<td>Name</td>
<td>Administrator</td>
<td>Nearcast center</td>
</tr>
<tr>
<td></td>
<td>Activity</td>
<td>Administrator / External system</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Nearby devices</td>
<td>Nearcast center</td>
<td>Nearcast server</td>
</tr>
<tr>
<td>Device</td>
<td>Address</td>
<td>Device</td>
<td>Nearcast center</td>
</tr>
<tr>
<td></td>
<td>Manufacturer</td>
<td>Device</td>
<td>Nearcast center</td>
</tr>
<tr>
<td></td>
<td>Model</td>
<td>Device</td>
<td>Nearcast center</td>
</tr>
<tr>
<td></td>
<td>Operating System</td>
<td>Device</td>
<td>Nearcast center</td>
</tr>
<tr>
<td></td>
<td>Hardware capabilities</td>
<td>Device / Device database</td>
<td>Nearcast center</td>
</tr>
<tr>
<td>Content</td>
<td>Name</td>
<td>Administrator / External system</td>
<td>Nearcast server</td>
</tr>
<tr>
<td></td>
<td>Transactions</td>
<td>Nearcast center</td>
<td>Nearcast server</td>
</tr>
</tbody>
</table>

also shows the different sources of context in the nearcast framework. A first source is the nearcast application, which is able to determine the user’s contact list, calendar items and the user may configure for instance his name and current mood. Second, the nearcast center can determine the location of users, their transaction history and their social situation. Third, context can be configured, maintained or imported by the administrators of the system. They should configure all stages by setting its name and activity for instance. Fourth, the nearcast framework may query an external system via internet to determine a specific type of context information.
For example, the schedule of the festival can be maintained in Google Calendar, which is queried by the nearcast framework using web services. In this way, the nearcast framework is able to determine what performances are playing at the moment and what performances are coming up.

The last column in 表格 未找到引用源。 shows on which hardware component the context components are allocated. This is relevant for analyzing the performance of the system, because the allocation determines what communication must take place. To illustrate this, the contact list service is used as an example. This service runs at the nearcast client and presents the contact list to the users. It requests the nearcast center for the location of all users in the contact list. This request is passed to the nearcast server. Now, there are two possibilities: either the location components run on the nearcast server or the location components run on the nearcast centers.

First, a scenario is studied when the location component runs on the nearcast center. Each time a user changes location, the corresponding sensor updates the connector component. When this component runs on the nearcast center, the location information is consequently also stored on the nearcast center. When a nearcast application requests the information needed to build the contact list, it first sends a request to the nearcast center that passes this request to the nearcast server. The nearcast server should query all nearcast centers to locate the devices in the contact list and request the necessary information.

In the other approach, the location component runs on the nearcast server. In this case, the user’s location is stored on the nearcast server. A nearcast client’s request can be replied directly, without querying other nearcast centers. This reduces the delay to reply significantly.

Another aspect is the amount of generated traffic. In the first approach, data is transferred from the nearcast centers to the nearcast server only when the nearcast application has requested the context information or an update of context information is sent to a subscribed nearcast application. The second approach generates more traffic, because every time the context information changes, it is sent to the nearcast server.

On the other hand, once a nearcast center has become unreachable because it is disconnected for example, this information becomes unavailable.

The decision to allocate a context information component on the nearcast server or nearcast center is thus a tradeoff between how fast the context information is needed and how much data traffic the context information generates. In fact, the user’s location is a special case, because when the user’s location is known, the nearcast server is able to directly address the nearcast center that contains the current context of a user. However, context information from other nearcast centers may also be needed.

Therefore, the allocation of each component needs to be a well-considered decision that realizes a good balance between the performance of the service and the network load. 表格 未找到引用源。 shows already the decisions for most types of context information.
7.4.3 **Behavior**

Figure 18 shows the sequence diagram of how a nearcast client application subscribes to location updates. First, the client sends a context subscription request to the nearcast center that passes the request to the location component on the nearcast server. This component returns the current location information. As soon as the location connector receives new information, an update is sent to the subscribed device immediately.

![Sequence diagram of context information updates](image)

7.4.4 **Relation with previous views**

The context view strongly relates to the deployment view, but also has relations with the layered decomposition view and the client server view. The layered decomposition view shows that all three software components (nearcast client, nearcast center and nearcast server) contain a context module. Actually, the context view describes this context module. An important part of the context view is how context components are allocated onto the different hardware components. The deployment view describes these hardware components and the relations between these components, such that performance of the nearcast framework regarding the context can be analyzed. The client-server view shows how the requests for context data flow through the nearcast framework.

7.4.5 **Architectural design decisions**

**AD14: use publish-subscribe pattern**

There are two approaches for exchanging context information between the components in the nearcast framework. In the first approach, clients can poll the server for updated context information. Designers should consider the delay of message delivery against the generated traffic. If a short delay is required for messages delivery, the client should poll often. However, this generates much traffic, which is not desirable. The second approach uses the publish-subscribe pattern, in which components subscribe for specific events, changes in the context for example. Traffic between the subscriber and publisher and the delay of a message delivery are both minimized. The condition to apply this pattern is that the publisher is able to call the subscriber, because the publisher must notify the subscriber. In a client-server architecture this is not always possible, because the server cannot address its clients. However, the nearcast framework does provide this functionality with the persistent connections.
AD15: Use a hybrid model to store context information
AD16: Allocate user location at the nearcast server

The choice for a context architecture model is of great influence on the quality of the nearcast framework, because context-awareness is one of its main aspects. A context architecture model determines how context is gathered, represented and stored, how context can be used and how the system can reason about context. These aspects have also impact on the infrastructure of the nearcast framework.

The four different context architecture models, presented in section 4.6, are assessed on their suitability for the nearcast architecture, using a quality assessment for the quality attributes that are important for the nearcast architecture: flexibility and performance (Table 19). Reliability could not be evaluated because the related quality scenarios are not affected directly by these models.

Table 19 - Comparison of quality attributes for available context models.

<table>
<thead>
<tr>
<th></th>
<th>Blackboard</th>
<th>Service Infrastructure</th>
<th>Context Widgets</th>
<th>COOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Reliability</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Performance</td>
<td>+/-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**Blackboard**

The blackboard model is rather flexible because all context information is stored at one place. This eases the access to context information for new services or external systems: these only need to know how to access the repository to retrieve the needed context information. Context is stored in one large repository at the nearcast server. Context-aware services are able to query this repository and subscribe to be updated when changes occur to this context. Different types of context data can be aggregated easily enabling the fast context lookup for services. These have a positive influence on the performance. In addition, communication is relatively easy, because all nearcast centers only have to communicate with the nearcast server. However, the network will be loaded heavily when sending lots of context information to the nearcast server. This influences performance negatively. Therefore, the performance of the blackboard model is marked with +/-.

**Service infrastructure**

The service infrastructure stores the context information at the distributed components, e.g. the nearcast center’s context would be stored at the nearcast center itself and can be requested via web services. The mobile phone’s context, including the user’s context, should be stored at the mobile phone itself and can be requested via a nearcast center using Bluetooth, for example. Services running on the same entity could obviously access the context information directly. A service infrastructure-based nearcast framework will therefore have good impact on flexibility. New sensors and services can be added without affecting the already existing ones.

When a nearcast user wants to know the status of all of its contacts, the nearcast center has to retrieve the status of all these devices from all other nearcast centers. Once the nearcast center has gathered all information, it should determine the status of all devices. The communication with all nearcast centers causes a delay in the nearcast application’s request, which is negative for the performance of the system. An incidental disadvantage is that once devices disappear from the network, their context information
also becomes unavailable. In some situations, this could be awkward. For example, when a service needs the transaction history of a specific device, but the nearcast center that contains this information disappeared from the network.

**Context Widgets and COOL**
The Context Toolkit and the COOL architecture have rather the same ideas about the representation of context information. Both models use a context model that focus on the representation of context information with sensors, aggregators, services, etc. However, while components in the COOL architecture can represent multiple types of context information, widgets in the Context Toolkit represent only one type of context information. The advantage of the context widgets is that there is a clear one-to-one mapping of context information on objects. A COOL component cannot be directly associated with one kind of context information, but it describes what kinds of context information it can provide in the Object Repository. Another difference is that COOL components are more autonomous as they are able to provide and execute services, while the Context Toolkit only allows the services objects to execute services.

To determine the available context information, new services or external systems can access the Object Repository in the COOL framework or the Discoverer in the widget approach. These new services are autonomous components that run outside the nearcast framework and should be added at run-time. A nearcast framework based on either widgets or COOL components will be therefore very flexible.

The context representation is independent from the way context information is stored. For developers it is transparent where context information comes from or where it is stored. This enables a hybrid solution for the storage of context information. Context information that should be always available can be stored at the nearcast server while context information that requires less availability can be stored at a nearcast center or mobile device. Data exchange between the components within the framework can be reduced in this way, allowing a good balance between performance and availability. This concept is positive for the performance of the system.

**Conclusions**
Both the COOL and widget model are the most suitable approaches for gathering and representing context information, because they provide both high flexibility and performance. Because in the future the nearcast framework might be integrated with other applications based on the COOL framework, the COOL model will form the foundation for the nearcast architecture.
8 Validation

Validation of software architecture is part of the software development process and results in a number of benefits. First, software development costs can be reduced. The earlier deficiencies in the architecture are found, the better the development process can adapt to these changes and the lower the costs. Second, by evaluating software architecture, design decisions and their rationales are explicitly registered, which is invaluable information for later use. Third, validation of requirements uncovers risks, conflicts and tradeoffs. In short, architectural validation tends to increase quality, control costs, and decrease budget risk.

Bass et al. (2003) describe several ways to validate software architecture. First, the Architecture Tradeoff Analysis Method (ATAM) and the Cost-Benefit Analysis Method (CBAM) are questioning techniques that use scenarios as a vehicle for asking probing questions about how the architecture under review responds to various situations. Second, measuring techniques rely on all kinds of quantitative measurements. For example, after building a simulation or prototype, different runtime aspects of the system, such as performance or availability, can be measured. Alternatively, a mathematical model can be used to calculate the qualities of a system.

In this thesis, both kinds of techniques are applied. First, methods from ATAM are used to analyze the architecture in order to determine how the architectural design decisions affect the various quality attributes. These decisions are called sensitivity points. Architectural decisions that affect multiple quality attributes are identified as tradeoff points. In addition, risks are defined to indicate which architectural decisions may lead to undesirable consequences with respect to the stated quality attribute requirements. These methods help to provide more insight into the strengths and weaknesses of the software architecture.

Second, a prototype is built to help to perform an analysis of the driving quality attributes flexibility, reliability and performance. With a prototype, practical problems may be encountered that can help in the design decision process (Sommerville, 2001). The prototype is also built to explore the technical possibilities and restrictions of the use of Bluetooth.

The remainder of this chapter describes the following aspects. Section 8.1 discusses the implementation of the prototype. Section 8.2 discusses the implemented functionality. Section 8.3 presents a quality attribute tree and analyzes each quality scenarios to discover risks and sensitivity and tradeoff points. Further, each quality scenario is evaluated using the prototype. Section 8.4 discusses the results and concludes this chapter.

8.1 Implementation

A prototype of the nearcast framework is implemented following the software architecture described in Chapter 7. This section discusses how the components of the nearcast framework, i.e. the nearcast center, the nearcast client, the nearcast server and the administration interface, are developed and what development environments are chosen.
8.1.1 Nearcast center
The nearcast center is the server component in the client-server interaction between the mobile device and the nearcast framework. Several software elements are needed to interface the nearcast center application with the physical layer (Figure 19). The Bluetooth stack is responsible for the implementation of the Bluetooth wireless standards specification. A number of implementations of the Bluetooth stack are available (Auletta, 2006). BlueZ [8], the official Bluetooth stack implementation for Linux, is the only free implementation and therefore chosen to be used in the nearcast framework.

![Diagram showing the interaction between applications and physical layer on the nearcast center](image)

On top of this Bluetooth stack, applications need an interface to access the Bluetooth functionality. The nearcast center application is developed in Python. This programming language is very suitable to develop prototypes or proof-of-concepts because of its steep learning curve. Furthermore, (part of) the application code of the nearcast center can be ported to the nearcast client and visa versa, as discussed in the next section. PyBlueZ [4], a Python extension module written in C, provides access to system Bluetooth resources. This extension module is used to control the Bluetooth adapter.

8.1.2 Nearcast client
The nearcast client is implemented in Python for Series 60 (PyS60) [6]. Nokia ported the programming language Python to the Series 60 mobile device. PyS60 offers a wide range of functionality such as handling the on-board camera, providing access to contacts and calendar items, sound recording and playback, providing access to system information such as the IMEI number, disk space, free memory, and most importantly, access to the Bluetooth API.

PyS60 is chosen in favor of other alternatives such as J2ME or Windows Mobile because this development environment enables the rapid application development. Further, its Bluetooth API supports the most important Bluetooth functions and the author of this thesis had experience with developing Bluetooth applications in PyS60.

8.1.3 Nearcast server
The nearcast server is also implemented in Python, such that source code can be exchanged and re-used between the nearcast center and nearcast server. The nearcast server also provides the administration interface, which is implemented as a web-based application that runs on an Apache2 web server. This administration interface is written in PHP and uses a MySQL database to store information.

8.1.4 Hardware setup
To run some experiments the following test-bed has been setup.
- **Nearcast center and server**: Dell OptiPlex GX 270, Intel Pentium 3 processor (800MHz), 512 MB RAM, equipped with two Belkin Bluetooth 2.0 adapters, connected to the internet, running Debian Linux.

- **Nearcast center**: Dell OptiPlex SX 270, Intel Pentium 3 processor (800MHz), 512 MB RAM, equipped with two Belkin Bluetooth 2.0 adapters, connected to the internet, running Debian Linux.

- **Nearcast client**: Nokia N70 smartphone with Symbian OS v8.1a, S60 2nd edition, Feature Pack 3, running Python for Series 60, Bluetooth support

The nearcast center component runs on two different machines, while one of these machines also functions as nearcast server and runs the administration interface. The nearcast client component, obviously, runs on the mobile device.

### 8.2 Functionality

This section discusses what functionality the prototype contains, why these use cases are elicited and how they are represented in the prototype. The prototype does not capture all functionality, but focuses on architectural significant use cases and quality attribute requirements. Some of the use cases are completely implemented, some of them are only partly and some of them are not implemented (Table 20).

The download and upload content use cases, and content push and pull use cases are completely implemented, because these are architectural significant use cases. If users are not able to download or upload info, or the system is not able to push or pull content, then there is no nearcasting. The prototype supports all these use cases, because using all communication primitives identified in section 3.1.2 is one of the unique points of the nearcast framework.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Description</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR 1</td>
<td>Download application</td>
<td>Partly</td>
</tr>
<tr>
<td>FR 2</td>
<td>Download content</td>
<td>Yes</td>
</tr>
<tr>
<td>FR 3</td>
<td>Upload content</td>
<td>Yes</td>
</tr>
<tr>
<td>FR 4</td>
<td>Content push</td>
<td>Yes</td>
</tr>
<tr>
<td>FR 5</td>
<td>Content pull</td>
<td>Yes</td>
</tr>
<tr>
<td>FR 6</td>
<td>Subscribe services</td>
<td>Partly</td>
</tr>
<tr>
<td>FR 7</td>
<td>Configure nearcast services</td>
<td>No</td>
</tr>
<tr>
<td>FR 8</td>
<td>Upload create or modify content</td>
<td>No</td>
</tr>
<tr>
<td>FR 9</td>
<td>View statistics</td>
<td>Partly</td>
</tr>
<tr>
<td>FR 10</td>
<td>Deploy and configure nearcast system</td>
<td>Partly</td>
</tr>
<tr>
<td>FR 11</td>
<td>View and modify privileges</td>
<td>No</td>
</tr>
<tr>
<td>FR 12</td>
<td>Create and deploy mobile nearcast application</td>
<td>No</td>
</tr>
<tr>
<td>FR 13</td>
<td>Create new service</td>
<td>No</td>
</tr>
</tbody>
</table>

The remainder of this section discusses how these use cases are represented in the prototype. First, users are able to download content as the prototype application offers the option to download an image. Second, users are able to upload content, as they are able to choose one of the pictures from their mobile phone to upload. Third, the nearcast
framework is able to push content to the users, as users are able to send messages to each other that are delivered by the nearcast framework. Fourth, the nearcast framework is also able to pull content from the nearcast client as the nearcast framework can request information about the sent messages.

Further, there are a number of use cases partly implemented. The Download application use case is implemented in an earlier version of the prototype, but due to incompatibilities omitted from the current prototype. The Subscribe services use case is implemented in the way that as soon as the user opens the contact list, the nearcast client automatically subscribes to location updates of these users. However, users are not able to subscribe explicitly to a service via neither a website nor the application itself. The Deploy and configure nearcast framework is also partly implemented: service providers are able to view which nearcast centers are currently connected. Finally, for the View statistics use case, content providers are able to view which users have been close to the nearcast center for the last hour.

The other use cases were not implemented because they are not architectural significant use cases and the available amount of time was limited.

This section has presented which use cases have been implemented, which partly or not, and how these are represented in the prototype. Although not all use cases are implemented, the prototype can be used to evaluate the software architecture because the architectural significant use cases have been implemented.

8.3 Quality attributes

This section studies the quality attributes of nearcast framework’s software architecture by performing a theoretical and practical validation. The theoretical validation follows the ATAM (Kazman et al. 2000), which includes the generation of a quality attribute utility tree (section 8.3.1) and an analysis of the quality scenarios to identify the sensitivity points, tradeoff points and risks (section 8.3.2). As well as this theoretical analysis, each scenario is also evaluated using the prototype (also in section 8.3.2).

8.3.1 Quality attribute utility tree

Before the software architecture can be evaluated, the quality attribute goals have to be specifically defined. Therefore, a quality attribute utility tree is created. A quality attribute tree begins with utility as the root node, which is an expression of the overall goodness of the system. Quality attributes form the second level because these are the components of utility. Because these quality attributes are rather vague, they are refined with attribute refinements. These refinements are accompanied with one or more quality attribute scenarios, which are concrete enough to be tested and analyzed.

Table 21 presents the quality attribute utility tree for the nearcast framework. There is a great similarity with the quality attribute requirements from Chapter 6. The scenarios from this table form the foundation for the evaluation.

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Attribute refinement</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>Flexibility for new services</td>
<td>The developers want to add a new service to the nearcast framework. The changes will be made to the code at design time.</td>
</tr>
<tr>
<td>Quality Attribute</td>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Integration with external systems</td>
<td>An external system’s developer wants to integrate the nearcast system in its own system. The service provider adds an account for this system, after which the external system’s developer is able to access the nearcast framework according to the specified interface. The nearcast system does not need to be modified or restarted.</td>
<td></td>
</tr>
<tr>
<td>Communication technology independence</td>
<td>The owner of the nearcast framework wants to replace Bluetooth with WiFi. The developers only need to change the communication layer modules of the nearcast client and nearcast center.</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Users initiate 360 transactions per minute stochastically on the nearcast center that is operating normally; 99% of these requests arrive at the nearcast center correctly.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The nearcast center continuously scans for nearby devices in its environment and should find 99% of the devices within 15 seconds under all circumstances.</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>A nearcast client enters the range of a nearcast center. The nearcast center is already connected to 30 devices. The new device is connected within 5 seconds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A user sends a message to another user who is located at another nearcast center. Under normal operations, messages are delivered within 5 seconds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30 users are simultaneously downloading content from the nearcast center. The throughput of each connection is at least 70 Kb/sec.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The content provider wants the system to be able to handle 50,000 visitors simultaneously. The nearcast system is extended in such way that it has this capacity, with no decreased qualities.</td>
<td></td>
</tr>
</tbody>
</table>

### 8.3.2 Scenarios

In this section, each quality scenario from the quality attribute utility tree is analyzed to assess how well the architecture supports the scenario. For each quality scenario, the architectural decisions are identified as sensitivity point, tradeoff point and/or risk. An architectural decision is a sensitivity point when the quality scenario is sensitive to this architectural decision, i.e. if this decision has influence on achieving the quality attribute response in question. When an architectural decision is a sensitivity point for more than one quality attribute, it is identified as tradeoff point. Further, when it is not clear whether a decision will achieve its goal or when the sensitivity of a decision is rather high, it is marked as risk.

The remainder of this section presents an architectural approach analysis for each quality scenario, including a motivation why the architectural decisions are identified as sensitivity point and/or risk. Further, the prototype is evaluated in how it influences the quality scenario in question. The sensitivity points are discussed in section 8.4.
Scenario 1: Easily develop new services

Table 22 shows the architectural approach analysis for scenario 1. The nearcast framework’s components are designed according to a three-layer architecture. When developing a new service, only the two upper layers, the presentation and application layer have to be modified, while the lowest layer, the communication layer is left unmodified. In this way, changes to the software are kept local. The layered architecture also prevents the ripple effect: changes to one layer do not lead to changes in adjacent layers. Thus, the flexibility in developing new services for the nearcast framework is sensitive to the choice for a layered architecture.

Table 22 - Architectural approach analysis for scenario 1

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Easily develop new service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operation, design time</td>
</tr>
<tr>
<td>Stimulus</td>
<td>The developers want to create a new service</td>
</tr>
<tr>
<td>Response</td>
<td>The changes will be made to the code at design time and will affect no more than six software modules.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD0. Layered decomposition</td>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD2. Use nearcast server for communication</td>
<td>S2</td>
<td>T1</td>
<td></td>
</tr>
<tr>
<td>AD3. Message-based communication</td>
<td>S3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The use of a central server has a positive effect on this quality scenario, because a central server minimizes the number of modules that consume or produce data. A component can access the nearcast server for all information, while in an architecture with no central server, components should access multiple nearcast centers to retrieve their information.

A third architectural decision that influences the flexibility in developing new services is the decision to use message-based communication. The protocol can be reused for context-related functions or can be extended easily for new services.

Prototype evaluation

To evaluate this scenario for the prototype, the following analysis will determine how many modules have to be changed when adding a new service. Some regular visitors of the festival were asked to think a new service. One idea was to provide waiting times for the queues at the entrance of the festival, such that visitors on the campsite can check the best moment to enter the festival terrain.

To implement this service in the nearcast framework, first, a waiting times sensor and component are added to the nearcast server. The sensor retrieves the queue information, while the component transforms this information into suitable data for the nearcast framework and enables other components to subscribe to updates.

The nearcast client needs the most modifications: the user interface needs to be able to display the queue information, the dispatcher needs to handle the incoming and outgoing messages to request or retrieve the queue information, and the context module needs to store and represent the queue information.
The nearcast center does not need to be modified, because requests from the nearcast client can be passed directly to the nearcast server using general context information functions. This also means that the interfaces between the components stay the same.

Thus, two components need to be added to the nearcast server, three components need to be changed on the nearcast client and no modifications to the interfaces have to be made. A total number of 5 modules needs to be changed, which is less than required.

**Scenario 2: Integration with external systems**

Table 23 shows the analysis for scenario 2. A first architectural decision that influences the integration with external systems is the use of a central server for communication. This has a positive effect on the interoperability, because external systems only have to communicate with this central server and do not need to have any knowledge about the nearcast system’s internal communication.

<table>
<thead>
<tr>
<th>Scenario 2</th>
<th>Easily integrate with external systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operation, run time</td>
</tr>
<tr>
<td>Stimulus</td>
<td>An external system’s developer wants to integrate the nearcast system in its own system.</td>
</tr>
<tr>
<td>Response</td>
<td>The service provider adds an account for this system, after which the external system’s developer is able to access the nearcast framework according to the specified interface. The nearcast system does not need to be modified or restarted.</td>
</tr>
</tbody>
</table>

Other decisions that have a positive effect on the integration with external systems are the choices for XML and message-based communication. XML is a generic communication language that has a widespread usage, resulting in high interoperability. The interfaces are small and simple and have a clear goal and result. External systems do not need to have a specific remote computing environment installed, but only need to be able to send and receive XML messages over a TCP or socket connection.

A fourth decision that has effect on the integration with external systems is the requirement for persistent connections. Although performance will be higher with such connections, because events are delivered immediately at the external system instead of the external system polling for new events, persistent connections are more complex due to the two-way communication. This can result in integration problems causing a lower interoperability. Therefore, this sensitivity point is marked as a risk.

**Prototype evaluation**

Unfortunately, the prototype not integrated in an external system. Therefore, this scenario does not have a prototype evaluation.
Scenario 3: Replace nearcast technology

Table 24 shows the architectural analysis for the third quality scenario: replacing the nearcast technology. First, by using a layered architecture, Bluetooth-specific code is isolated in the communication layer, specifically in the client and center interface modules. Therefore, replacing the nearcast technology will only affect these modules.

Table 24 - Architectural approach analysis for scenario 3

<table>
<thead>
<tr>
<th>Scenario 3</th>
<th>Replace nearcast technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Environment</td>
<td>Deploy time</td>
</tr>
<tr>
<td>Stimulus</td>
<td>WiFi has become standard on mobile phone</td>
</tr>
<tr>
<td>Response</td>
<td>Communication layer has to be replaced</td>
</tr>
</tbody>
</table>

Second, replacing the nearcast technology will have also impact on other qualities of the nearcast system, in particular performance, because the different communication technologies all have very different characteristics, such as bandwidth, connecting time, etc.

Third, the decision to use XML for communication also affects the flexibility in replacing the nearcast technology. XML-based communication is suitable for almost each communication technology and does not require any modification in other layers.

Prototype evaluation

Unfortunately, the prototype is not used to replace Bluetooth with another technology. Therefore, this scenario does not have a prototype evaluation.

Scenario 4: Connection robustness

Table 25 shows the results of the analysis of connection robustness in the nearcast framework. The connection robustness is sensitive to the decision to use Bluetooth. However, characteristics for other nearcast technologies are not available, thus no statements can be made if connection robustness becomes better or worse with other technologies. It is therefore marked as a risk.

Table 25 - Architectural approach analysis for scenario 4

<table>
<thead>
<tr>
<th>Scenario 4</th>
<th>Connection robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Reliability</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operation</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Users initiate 360 requests per minute stochastically on the nearcast center</td>
</tr>
<tr>
<td>Response</td>
<td>99% of these requests arrive at the nearcast center correctly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD2. Use Bluetooth as nearcast technology</td>
<td>S11</td>
<td></td>
<td>R1</td>
</tr>
<tr>
<td>AD6. Nearcast center connects to the nearcast client</td>
<td>S12</td>
<td>T2</td>
<td></td>
</tr>
</tbody>
</table>
A second decision is to let the nearcast center connect to the nearcast client. In this way, the nearcast client depends on the nearcast center whether there is a connection available or not, which means that the connection robustness partly depends on the discovery reliability. Assuming that the software architecture supports the quality scenario of the discovery reliability, this does not lead to a risk.

A third design decision that has impact on the availability of the nearcast center is the use of persistent connections. The longer connections are used, the more often these connections will break, resulting in a decreased availability of the nearcast center. This decision is marked as a risk.

Prototype evaluation
Experiments have shown that the prototype does not meet the requirements of this quality attribute scenario. The prototype is able to handle one client, but the more nearcast clients, the more requests do not arrive at the nearcast center or nearcast centers’ responses do not arrive at the nearcast client, resulting in lower connection robustness.

A part of these failures is caused by the fact that the prototype does not take into account that the Bluetooth connection breaks when it has been idle for about one minute. The Bluetooth specification mentions the link supervision timer that detects link losses. Each time the device receives a packet, the timer is reset, but when this timer reaches the specified value, the connection is considered to be disconnected. This could be the reason that connections break down after a certain period. However, a further analysis of this problem is required to find the exact reason. Because the decision to use persistent connections is related to this problem, it is marked as a risk.

Scenario 5: Discovery reliability
Table 26 shows the defined sensitivity and tradeoff points for the discovery reliability scenario. This scenario, like the other reliability and performance scenarios, also depends on the choice of Bluetooth because this technology is used to discover devices.

<table>
<thead>
<tr>
<th>Scenario 5</th>
<th>Discovery reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Reliability</td>
</tr>
<tr>
<td>Environment</td>
<td>All circumstances, normal operation</td>
</tr>
<tr>
<td>Stimulus</td>
<td>A nearcast component needs to connect to another nearcast component</td>
</tr>
<tr>
<td>Response</td>
<td>The discovery should report 99% of the nearby devices</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD2. Use Bluetooth as nearcast technology</td>
<td>S14</td>
<td>Tradeoff</td>
<td>R1</td>
</tr>
<tr>
<td>AD6. Nearcast center connects to the nearcast client</td>
<td>S15</td>
<td>T2</td>
<td></td>
</tr>
<tr>
<td>AD9. Run Bluetooth discovery for 15.38 seconds</td>
<td>S16</td>
<td>T4</td>
<td></td>
</tr>
</tbody>
</table>

The decision to let nearcast centers connect to nearcast clients also affects the discovery reliability, because in this case the nearcast center has to perform a Bluetooth discovery to find the nearcast devices. In the other case, nearcast clients connect to the nearcast center, which might result in another discovery reliability.
Further, this scenario is sensitive to the length of the Bluetooth discovery. The longer the discovery, the higher the chance that it has found every device. One factor cannot be excluded from this quality attribute, which is the fact that devices that are performing a discovery, cannot be found by a nearcast center or any other Bluetooth device.

Prototype evaluation

The prototype has shown that it meets the requirements for this quality attribute scenario. All devices are discovered correctly, up to around 12 devices, including 8 nearcast devices. The current prototype has not been tested in an environment with more than twelve devices.

However, the prototype uncovers one problem with the device and service discovery: it does happen that, after a connection is broken incorrectly, the nearcast center is not able to discover the services of a mobile device anymore. Only rebooting the mobile device enables the nearcast center to find the device’s services again. To solve this problem, further analysis and testing is required to determine when and why connections break and how this can be avoided.

Scenario 6: Connecting time

Table 27 shows the analysis of the architectural approaches that affect the connecting time scenario. First, the choice for Bluetooth has a great effect on the connecting time, because the mechanisms of this technology determine how to and how long it takes to connect between two devices.

<table>
<thead>
<tr>
<th>Scenario 6</th>
<th>Connecting time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Performance</td>
</tr>
<tr>
<td>Environment</td>
<td>Thirty devices around the nearcast center, all connected</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Another nearcast user enters the range of the nearcast center</td>
</tr>
<tr>
<td>Response</td>
<td>The new nearcast client should be connected with the nearcast center within 5 seconds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD2. Use Bluetooth as nearcast technology</td>
<td>S17</td>
<td></td>
<td>R1</td>
</tr>
<tr>
<td>AD6. Nearcast center connects to nearcast client</td>
<td>S18</td>
<td>T2</td>
<td>R3</td>
</tr>
<tr>
<td>AD7. Use Bluetooth service discovery</td>
<td>S19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD9. Let Bluetooth discovery run for 15.36 seconds</td>
<td>S20</td>
<td>T4</td>
<td></td>
</tr>
<tr>
<td>AD10. Bluetooth discovery returns devices one by one</td>
<td>S21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD11. Separate device and service discovery</td>
<td>S22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second, the decision to let the nearcast center connect to the nearcast clients also has great influence on the connecting time. When there are thirty devices around the nearcast center all waiting to be connected, it depends on the configuration and performance of the nearcast center how long it takes before all devices are connected.

In the alternative, to let the client connect to a nearcast center, the nearcast client performs a discovery or connects to the nearcast center based on cached addresses. This results in a more predictive connecting time, because the connecting time only depends on the performance of the nearcast client.
The use of the Bluetooth service discovery protocol also has an effect on the connecting time: using registered addresses only may result in a lower connecting time, but the nearcast center will not connect to non-registered devices, unless it performs a service discovery on these devices.

The connecting time is also sensitive to the discovery time. The longer the Bluetooth discovery takes, the less often devices are reported. This means that the nearcast center less often tries to connect to these devices, which results in a longer connecting time.

The architectural design decisions AD10 and AD11 have been taken to decrease this connecting time. The decision to return the devices one by one in the discovery process has a positive effect on the connecting time, because the sooner a device is reported, the sooner the nearcast center is able to connect to this device. Separating the device and service discovery results in shorter connecting times, because these processes can be executed in parallel.

Prototype evaluation
Experiments with the prototype have shown that the decision to let the nearcast center connect to the nearcast client has high impact on the connecting time. The nearcast center has to perform a service discovery on all unknown devices that potentially could have the nearcast client installed. The length of the service discovery process varies from 0.5 to 20 seconds. Separating this process from the device discovery already led to a shorter connecting time. In an environment with few devices, the connecting time will be low enough. However, as soon as there appear eight or more devices that all want to connect, the connecting time becomes too high. Therefore, the decision to let the nearcast center connect to the nearcast client is marked as a risk.

The main problem will be a situation where many devices nearly simultaneously enter a nearcast center’s range and the nearcast center has to perform multiple service discoveries. When these service discoveries are performed subsequently, as in the current prototype, they will take too much time to support the quality requirements. Therefore, the duration and organization of the service discovery should be studied further. Possible solutions are to run the service discovery in parallel, on either one Bluetooth adapter or multiple adapters. Unfortunately, the current version of the PyBlueZ API does not support the individual addressing of different Bluetooth adapters.

Scenario 7: User sends a message to another user
Table 28 shows the results for the analysis of scenario 7. Obviously, the choice for Bluetooth has an effect on the network latency, because this latency depends on the communication technology used.

<table>
<thead>
<tr>
<th>Scenario 7</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Performance</td>
</tr>
<tr>
<td>Environment</td>
<td>Many devices around nearcast center</td>
</tr>
<tr>
<td>Stimulus</td>
<td>A user sends a message to another user who is located at another nearcast center</td>
</tr>
<tr>
<td>Response</td>
<td>The message is delivered at the other user within 5 seconds.</td>
</tr>
<tr>
<td>Architectural decision</td>
<td></td>
</tr>
</tbody>
</table>
AD2. Use Bluetooth as nearcast technology  S23  R1
AD3. Use nearcast server for communication  S24  T1
AD8. Asynchronous communication  S25  T3
AD13. Use persistent connections  S26  T5
AD14. Use publish-subscribe pattern  S27
AD16. Allocate user location on nearcast server  S28

A second design decision that has effect on the network latency is the use of a central nearcast server for communication. This requires two extra hops in the message delivery process: the nearcast center should send the request to the nearcast server, which in turn should send the request to the other nearcast center. Without central server, the nearcast center can directly deliver the request at the other nearcast center, resulting in a lower latency of message delivery. The nearcast centers should keep an administration of addresses of other nearcast centers, such that all nearcast centers can connect to all other nearcast centers to exchange information.

Third, asynchronous communication leads to a lower latency because messages can be forwarded directly without having to wait on other data transfers.

Fourth, the use of persistent connections results in a lower latency, because devices do not need to setup a connection.

Fifth, the publish-subscribe pattern also results in a lower latency, because subscribed components are immediately informed about new messages, context updates, etc.

Sixth, the allocation of the user’s location onto the nearcast server may help to decrease the network latency in the nearcast framework, since messages that arrive on the nearcast server and have a specific device as destination, can be directly passed to the relevant nearcast center. When the nearcast server needs to determine the device’s location first, the network latency will be higher.

 Prototype evaluation
The experiments with the prototype show that the messages are delivered in less than 5 seconds. However, sometimes messages are not delivered because the connection robustness is not high enough, as evaluated in scenario 4. If messages from the nearcast client do not arrive at the nearcast center, the latency cannot be tested completely. The results of the messages that do arrive do support the quality requirement though.

 Scenario 8: User sends a photo to the nearcast center
Table 29 shows the results of the analysis of the architectural design decisions that are related to the throughput of connections between nearcast clients and nearcast centers. The first decision that has influence on this scenario is Bluetooth. This technology determines the maximum bandwidth of these connections.

<table>
<thead>
<tr>
<th>Scenario 8</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Performance</td>
</tr>
<tr>
<td>Environment</td>
<td>Peak operation</td>
</tr>
<tr>
<td>Stimulus</td>
<td>A user wants to upload a photo to the nearcast center</td>
</tr>
<tr>
<td>Response</td>
<td>The photo is uploaded with a data rate of 70 KB/sec</td>
</tr>
</tbody>
</table>
Asynchronous communication is negative for the throughput of one connection, because this connection is used for uploads and downloads. Simultaneously downloading and uploading content decreases the throughput of a connection.

Architectural design decision 12 states that connections are addressed on different adapters as much as possible. Because each Bluetooth adapter is able to use its full capacity, the maximum available bandwidth for each connection is ensured.

**Prototype evaluation**

The requirements state an expected data rate of 25.7 kilobyte per second. Experiments with the prototype showed a maximum data rate of around the 23 kilobyte per second with only one user. When adding more users, the data rate decreased rapidly. There are several possible causes and solutions for this low data rate.

1. Because the PyBlueZ API does not support OBEX connections, files are transferred between nearcast client and nearcast center using RFCOMM connections. A dedicated protocol was implemented to enable the sending of larger files over RFCOMM connections. A wrong implementation could be the reason for the lower data rates.

2. The nearcast software could also cause a lower expected data rate. There have been some difficulties with dealing with the Bluetooth sockets as earlier Bluetooth stack implementations on mobile phones only support one connection at the time. Therefore, one socket connection needs to be polled to check if there is any data ready to read from and should be kept free to be able to write data to it. Further, the server-side, the nearcast center, should be able to handle many devices simultaneously, but accessing the Bluetooth adapter in different threads resulted in Bluetooth device errors. More about these problems can be found in appendix E.

A second observation was that the stream of data stopped sometimes and continued again after some seconds or stopped definitely. The reason for this failure could not be found yet, but could be caused by the dedicated protocol.

Third, parsing XML messages containing larger objects takes more time than needed and to enable the inclusion in XML messages, these objects need to be encoded, resulting in larger messages. An alternative is to create multipart MIME messages [3], where objects are attached to the message with extra sections. In such situation, the multipart MIME message contains one section with the XML message and one for an image, for example. The image is included without having to encode it and the size of the MIME message will be smaller than when the image is encoded.

**Scenario 9: Scalability**

Table 30 presents the results of the scalability scenario analysis. The use of Bluetooth certainly has effect on the scalability of the nearcast framework, in particular the

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD2. Use Bluetooth as nearcast technology</td>
<td>S29</td>
<td></td>
<td>R1</td>
</tr>
<tr>
<td>AD8. Asynchronous communication</td>
<td>S30</td>
<td>T3</td>
<td></td>
</tr>
<tr>
<td>AD12. Connections are addressed on different Bluetooth adapters</td>
<td>S31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
nearcast center, because Bluetooth adapters have a limited capacity. According to the Bluetooth specification, a Bluetooth piconet may exist of eight devices, including the master device. Thus, one Bluetooth master device can handle up to seven connections simultaneously. This capacity is increased by installing more Bluetooth adapters.

Table 30 - Architectural approach analysis for scenario 9

<table>
<thead>
<tr>
<th>Scenario 9</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute(s)</td>
<td>Performance</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operations</td>
</tr>
<tr>
<td>Stimulus</td>
<td>A content provider wants to use the nearcast system for a much larger festival that requires a capacity for 50,000 users.</td>
</tr>
<tr>
<td>Response</td>
<td>The nearcast framework is extended with the necessary nearcast centers and each nearcast center is extended with a number of Bluetooth adapters. There are no modifications to the software.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural decision</th>
<th>Sensitivity</th>
<th>Tradeoff</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD2. Use Bluetooth as nearcast technology</td>
<td>S32</td>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>AD3. Use central server for communication</td>
<td>S33</td>
<td>T1</td>
<td>R5</td>
</tr>
<tr>
<td>AD6. Nearcast center connects to nearcast client</td>
<td>S34</td>
<td>T2</td>
<td>R3</td>
</tr>
<tr>
<td>AD13. Use persistent connections</td>
<td>S35</td>
<td>T5</td>
<td></td>
</tr>
<tr>
<td>AD14. Use publish-subscribe pattern</td>
<td>S36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD15. Hybrid model to store context information</td>
<td>S37</td>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>AD16. Allocate user location at nearcast server</td>
<td>S38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The use of a central server eases the communication, but when too many nearcast centers are connected to one nearcast server, the server will reach a point where it is not able anymore to handle all the requests. It is not clear with how many nearcast centers or how many users this situation occurs. The amount of traffic data and the load of the nearcast server should be analyzed to determine this breaking point. Therefore, this decision is marked as a risk.

The Bluetooth discovery process on the nearcast center has a limited capacity. The more devices the discovery has to process, the higher the connecting time will become. Therefore, the decision to let the nearcast center connect to the nearcast client is marked as a risk.

Moreover, each persistent connection continuously possesses one of the communication ports and thus decreases the nearcast center’s capacity.

The publish-subscribe pattern, the hybrid model to store context information and the allocation of user location on the nearcast server all have a positive influence on the scalability. These decisions result in less communication between the nearcast centers and nearcast server, which means that there can be more nearcast centers connected before the network is too much used. However, the hybrid context model lets designers choose where to store context information. It is possible to allocate context information such that it generates too much traffic. For example, if the designer wants to store all context information of the nearcast client devices on the nearcast server, this results in high traffic load. Therefore, this decision is marked as a risk.
Prototype evaluation

The capacity of the nearcast framework can be extended by either by adding a number of nearcast centers or plugging a number of extra Bluetooth adapters in a nearcast center. Adding a number of nearcast centers has not been tested in practice because no problems are expected by adding only a small number of nearcast centers. Before testing how the nearcast system reacts when adding a large number of nearcast centers, a more in-depth analysis should be made of the load of the nearcast server.

One of the nearcast centers has been tested with three Bluetooth adapters. One adapter is used for discovering devices, while the other two should be used for connections with the nearcast clients. Each adapter can handle seven devices at most and in order to use both adapters the test included eight devices. Unfortunately, not all eight devices could connect with the nearcast center. It is unclear whether this was caused by addressing the Bluetooth adapters wrongly, an incomplete service discovery or something else. However, some improvements to the nearcast center software have been made after this test. The results with three devices showed improvements in connection times, but the new version could not be tested anymore in a situation with a large number of devices.

It is thus desirable to be able to control the addressing of the available Bluetooth adapters in order to spread the connections over the available adapters. The current PyBlueZ API did not support this functionality.

8.4 Conclusions

This chapter evaluated the software architecture using a theoretical analysis and prototype. Section 8.2 motivated that the prototype of the nearcast framework supports the architectural significant use cases such that it can be used to validate the software architecture.

8.4.1 Results of theoretical validation

The architectural evaluation defined several sensitivity points, tradeoff points and risks. The sensitivity points are discussed in each scenario analysis, while below the tradeoff points and the risks are elaborated.

Tradeoff points

This section summarizes the identified tradeoff points and motivates why the current decision is maintained or a decision needs reconsideration.

T1. Use a central server for communication

The use of a nearcast server results in more flexibility in developing new services and a higher interoperability between the nearcast framework and external systems, but possibly in a higher network latency and scalability problems.

This network latency, however, will not be higher than allowed and to create a more scalable nearcast framework, the proxy pattern can be used (Bass et al., 2003). Therefore, the nearcast framework will use a central server.

T2. Nearcast center connects to nearcast client

The decision to let the nearcast center connect to the nearcast client results in lower power consumption of nearcast clients, but possibly in higher connecting times, lower discovery reliability and a worse scalability of the nearcast center.
In first instance, this thesis decided to let the nearcast center connect to the nearcast client. The architectural evaluation has resulted in a number of disadvantages of this method, such that this architectural design decision is reconsidered (see R2).

**T3. Asynchronous communication**

The asynchronous communication results in a lower latency (better), but also a lower throughput (worse), while synchronous communication results in a higher latency (worse) and a higher throughput (better).

In the case of synchronous communication, the nearcast center will not be able to notify the nearcast client while it is uploading a photo for example. The effect on the latency is much greater than the effect on the throughput. Therefore, the nearcast framework will use asynchronous communication.

**T4. Run Bluetooth discovery for 15.36 seconds**

The decision to let the Bluetooth discovery to run longer, i.e. 15.36 seconds instead of the usual 10.24 seconds, results in higher discovery reliability, but also higher connecting times.

The experiments of the prototype show that the discovery reliability is high enough with a discovery period of 15.36 seconds. Future designers may consider using a shorter discovery time (e.g. 10.24 seconds) in order to decrease the connecting time, but need to be sure that the discovery reliability stays high enough.

**T5. Persistent connections**

The use of persistent connections leads to lower network latency, but may also result in lower connection availability, possibly in a lower interoperability between the nearcast framework and external systems and less capacity of one nearcast center. R1 further elaborates the choice for persistent connections.

**Risks**

This section summarizes the identified risks and provides some suggestions to avoid these risks.

**R1. Use Bluetooth as nearcast technology**

One observation from the list of sensitivity points is that the choice of the nearcast technology influences all reliability and performance quality scenarios. The decision to use Bluetooth therefore has great influence on the qualities of the system. Because the effects of other technologies on the nearcast system are unknown, it is not clear whether this decision is negative or positive. Future work should study the effects of other technologies in detail to be able to make statements about the influence of the decision to use Bluetooth.

**R2. Use persistent connections**

The use of persistent connections may result in low interoperability because of the complexity of such connections. Moreover, experiments with the prototype showed that the persistent connections between the nearcast client and nearcast center sometime break resulting in lower connection robustness.
The use of persistent connections, especially between the nearcast client and nearcast center, should be reconsidered. The alternative is that both components create a connection only when they need to transfer information. This option was eliminated because connecting the nearcast client and nearcast center would cost too much time. Because it has become clear that the use of persistent connections also has some disadvantages, some tests with the prototype should be performed to assess detailed numbers about the connecting times and the number of times these connections fail. Tanenbaum (2002) discusses the HTTP/1.1 protocol that maintains connection for loading one web page because several elements must be downloaded for one such page. This concept could be used in the nearcast framework.

Further, now recent mobile phones are able to handle more than one connection, another alternative is to create two connections between the nearcast center and nearcast client: one that is controlled by the nearcast client and one for the nearcast center. Only the component that controls the connection is allowed to start using the connection, such that the other side is only listening. This may increase performance.

R3. **Nearcast center connects to nearcast client**
The decision to let the nearcast center connect to the nearcast client has impact on the connecting time. The nearcast center has to perform many discoveries, which may result in too large connecting times, in particular when increasing the capacity of the system.

Therefore, this decision is reconsidered here. The only reason to let the center connect to the client is that in a system that uses the alternative, let the nearcast client connect to the nearcast center, results in high energy consumption of the nearcast client. However, now several disadvantages of this decision have been identified, additional tests should be performed to determine exact numbers for discovery time for example. In addition, this process should be analyzed for improvements, like caching recently used nearcast centers.

R4. **Hybrid model to store context information**
Using a hybrid model to store context information means that context information has to be transferred to other places where it is stored initially. When increasing the system’s capacity, this may lead to network capacity problems if context information is allocated at unsuitable components such that lots of context has to be transferred.

Designers of the nearcast system should thus carefully decide where to store the different pieces of context. The architecture could include a guideline with suggestions where to store different types of context. For example, only the primary context information types (section 4.3) should be stored on the nearcast server and all secondary types of context information are stored locally.

R5. **Use of a central server**
The risk of using a central server is that in a system where much context information has to be transferred, the server gets too many requests to handle.
This capacity problem may be mitigated by applying a proxy design pattern: by introducing multiple nearcast servers and a load balancer that distributes the requests to the available nearcast servers, the capacity can be extended.

8.4.2 Results of the prototype

Table 31 shows the results of the validation of the prototype in short.

<table>
<thead>
<tr>
<th>Quality scenario</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR1. Flexibility in new services</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR2. Integration with external system</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR3. Replace nearcast technology</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR4. Connection robustness</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR5. Discovery reliability</td>
<td>Yes</td>
</tr>
<tr>
<td>QR6. Connecting time</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR7. Latency</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR8. Throughput</td>
<td>No, maximum of around 25 KB/sec for one device</td>
</tr>
<tr>
<td>QR9. Scalability</td>
<td>Tested not more than two nearcast centers, test to connect eight devices failed.</td>
</tr>
</tbody>
</table>

QR1-3. The first three quality scenarios have not been tested using the prototype.

QR4. The prototype did support the connection robustness scenario for a small number of devices. In a test with eight devices, connections were broken too often, but also with three nearcast clients, requests do not always arrive. One of the causes is that the Bluetooth connection breaks after it has been idle for some time.

QR5. The nearcast prototype did support the discovery reliability scenario, although the current version only has been tested in environments with up to twelve devices. Besides, when connections are broken incorrectly, the service discovery cannot always find the device’s services.

QR6. The connecting time scenario is only supported in environments with up to three devices. The connecting time may fluctuate much and depends strongly on the implementation of the nearcast center and the use of context information, i.e. using information about devices that have been discovered earlier. If it is decided to keep the nearcast center connecting to the nearcast clients, then the duration and organization of the device and service discovery should be studied further.

QR7. The latency scenario is supported as messages from one nearcast client are delivered at the other device within five seconds, if they are delivered. In some cases, messages get lost and do not arrive. This can be attributed to the lower connection robustness.

QR8. The nearcast system did not achieve the throughput scenario’s goal because the maximum transfer rate was around 23 KB/sec, even with one user. Two possible reasons for the lower than expected data rate are a wrong implementation of the dedicated protocol to support file transfers over a RFCOMM connection and a wrong implementation to listen and write to socket connections. These issues need further
research. Another observation is that file transfers sometimes stop, but continue after some seconds or stop definitely. A wrong implementation of the protocol could also be the reason for this issue. The size of messages can be decreased by omitting objects in the XML files and create multipart MIME messages.

**QR9.** Unfortunately, the scalability scenario was not supported because the prototype did not pass the test where eight devices tried to connect to the nearcast center. It is unclear whether this was caused by addressing the Bluetooth dongles wrongly, an incomplete service discovery or something else, thus further research is needed.

The results of the prototype show that there are several points for improvement, in particular the capacity of the nearcast center. This is the most important aspect that needs to further attention.

### 8.4.3 Development environment

The nearcast center prototype is developed in the programming language Python and uses a PyBlueZ to access the Bluetooth adapters. Although these tools are suitable to create a proof-of-concept, several limitations of the current version of PyBlueZ (0.9.1) have been identified:

1. The PyBlueZ API cannot access and control the different Bluetooth adapters directly.
2. The PyBlueZ API only supports RFCOMM connections, while actually OBEX connections are needed for file transfers. When the system is extended to distribute streaming audio and video, implementations for these Bluetooth profiles are needed also.
3. The PyBlueZ API does not support multicast.

Because of these limitations, the choice for the development environment is reconsidered here. Alternatives for Python and PyBluez are Java and a Java JSR-82 API or Microsoft’s .Net. Certainly, there are more development tools and support for these development environments. For example, recently Jellingspot [12] published their Jellingspot Data Server, which enables the development of individual and independent services that use Bluetooth to communicate with nearby devices. Therefore, future work should also include a study to other available development environments that at least overcome the limitations of the PyBlueZ API mentioned above.

Likewise, the nearcast client is developed in Python for Series 60. Although no serious limitations in PyS60’s Bluetooth API have been found, PyS60 application only run on Series 60 mobile phones. When creating a nearcast system that should be ready for production, there should be nearcast clients available that run on mobile phones other than Series 60. Because most mobile devices do support J2ME, future work should include the development of a J2ME nearcast client application.

Finally, a useful development tool may be Rococo's Impronto simulator [17]. This tool allows developers to test and configure applications before deploying them on Bluetooth devices.
9 Conclusions

This chapter concludes this thesis’ work by answering the research questions formulated in Chapter 1.

1. What are the requirements for a context-aware nearcast framework?
Before research question 1 can be answered, first sub questions 1.a. and 1.b. need to be answered. Therefore, these questions are discussed first followed by the answer on research question 1.

1.a. What are opportunities for innovative nearcasting services?
Chapter 5 compared the functionality and quality of existing nearcast systems and concluded with the following opportunities for innovative nearcast services:

- Support for all connection primitives: download content, upload content, content push and content pull;
- Support for streaming media;
- Include innovative methods for registration of new users and application provisioning;
- Use of different context information types to be able to improve current services or introduce new services.

This thesis focuses on the support for connection primitives and the use of different types of context information.

1.b. What are the benefits of context-aware features?
Chapter 6 performed an analysis of the benefits of context-aware features by discussing what context information and how this context information should be used. Some examples of the context-aware services are:

- The nearcast center knows the context of a discovered device and is thus able to decide which version of the nearcast application the user has to install.
- The nearcast center knows the context of the discovered devices and is thus able to decide whether to connect to each of these devices or not.
- The nearcast application knows the user’s context and is thus able to deliver services when the user finds is in a certain context.

Context-aware nearcasting thus enables content-providers to deliver unique and tailored content to the user, content that is adapted to the wishes and needs of the user. The better the system knows who is around, where they have been and what they like and dislike, the better the system can adapt its behavior. These are the benefits of the context-aware services of the nearcast framework.

As the previous sections have answered research questions 1.a. and 1.b., the following section discusses the answer on research question 1.

The requirements for the context-aware nearcast framework have been divided into functional and quality requirements. The support for the four connection primitives and the use of context information form the foundation for the functional requirements.
Flexibility, reliability and performance are the three driving quality requirements for the nearcast framework. Flexibility comprises the ability to develop new services easily, the interoperability with external systems and the ability to replace the nearcast technology. Reliability is determined by the discovery reliability and the connection robustness. Performance is measured by the connecting time, throughput, network latency and scalability.

2. What is the architectural design of a context-aware nearcast framework?
Chapter 7 presented the architectural design of the context-aware nearcast framework. The architecture focuses on a context-aware nearcast framework that supports the different connection primitives. The client-server view illustrates how the architecture supports the connection primitives, while the context view elaborates how the architecture handles context information. Further, the layered decomposition view shows how the system is decomposed of different components, the uses view shows dependencies between modules such that the modifiability can be estimated, and the deployment view shows the allocation of the software components on the hardware components and can be used to analyze performance. Each view is associated with a list of design decisions.

3. To what extent does the software architecture satisfy the quality requirements?
Chapter 8 validates the software architecture by performing a theoretical validation and a prototype validation.

Theoretical validation results
The theoretical validation resulted in a list of sensitivity points, tradeoff points and risks. The list of sensitivity points consists of all architectural decisions and is therefore omitted here. The tradeoff points and risks are listed below.

Tradeoff points
- T1. Use of a central server
- T2. Nearcast center connects to nearcast client
- T3. Asynchronous communication
- T4. Run Bluetooth discovery for 15.36 seconds
- T5. Persistent connections

Risks
- R1. Use Bluetooth as nearcast technology
- R2. Use persistent connections
- R3. Nearcast center connects to nearcast client
- R4. Hybrid model to store context information
- R5. Use of a central server

These tradeoff points and risks need further research to determine the most optimal solution. Chapter 8 already discussed suggestions for improvement.

Prototype validation results
The results of the evaluation of the prototype, summarized in Table 32, show that the prototype partly supports the requirements. The flexibility requirements have not been tested, while the connection robustness, connecting time and latency scenarios are only supported when the nearcast framework is operating with low capacity, i.e. around three devices.
Table 32 - Results from the prototype

<table>
<thead>
<tr>
<th>Quality scenario</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR1. Flexibility in new services</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR2. Integration with external system</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR3. Replace nearcast technology</td>
<td>Not tested</td>
</tr>
<tr>
<td>QR4. Connection robustness</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR5. Discovery reliability</td>
<td>Yes</td>
</tr>
<tr>
<td>QR6. Connecting time</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR7. Latency</td>
<td>Partly, for up to three devices</td>
</tr>
<tr>
<td>QR8. Throughput</td>
<td>No, maximum of around 25 KB/sec, for one device</td>
</tr>
<tr>
<td>QR9. Scalability</td>
<td>Tested not more than two nearcast centers, test to connect eight devices failed.</td>
</tr>
</tbody>
</table>

Section 8.4.2 discusses these results in detail. The main observation from the prototype validation is that increasing the capacity leads to problems in several scenarios. The scalability should be the most important issue when further developing the prototype.
10 Future work

Future work can be divided into three categories: for the software architecture, prototype and the nearcasting concept. Below, future work is enumerated for each category the issues for (in order of priority).

Future work for the software architecture

1. Study tradeoff points and risks. The results of the software architecture validation show that the prototype does not support all quality requirements. Future work should comprise of studying the identified risks and tradeoff points and using the prototype to test the quality scenarios to achieve full support for these quality requirements. The results should include statistical tests that support these conclusions.

Future work for prototype

1. Study alternative develop environment. The results of the validation showed that the Python development environment did not support all required functionality. Therefore, alternative environments like Java should be analyzed for if they do support this missing functionality.

2. Continue developing prototype. Further development of the prototype (either the current prototype or in a new development environment) is required to realize a full-working system. Chapter 9 has discussed the missing support for the requirements of the nearcast framework.

3. Study design issues of the nearcast center. If the current prototype is further developed, there are still some design issues for the nearcast center that should be solved (see appendix E). Future work should include the study of how these issues can be solved.

Future work for nearcasting concept

1. Streaming media. One of the opportunities for nearcast services was the use of streaming media. Future work could include a study for how Bluetooth supports the use of streaming media, in particular with the use of one of the Bluetooth profiles.

2. Device recognition. Section 5.3.5 has presented a method that can be used to identify mobile phones. Future work could include a study of if there are any other methods available and this method’s performance. The device recognition plays an important role in the distribution of applications.

3. Payment services. Another interesting feature of a nearcast framework that comes from one of the scenarios in section 3.2 would be that users are able to do payments. Future work should thus include a study of if the framework can be modified in such way that it is secure enough to enable payments.
A References

A.1 Articles


### A.2 Books


### A.3 Websites

13. JSON. http://www.json.org/
19. TNO. http://www.tno.nl

All referenced on December 17, 2006.
B  Bluetooth

Bluetooth wireless technology (Bluetooth, 2004) is a short-range communications technology intended to replace the cables connecting portable and/or fixed devices while maintaining high levels of security. The key features of Bluetooth technology are robustness, low power, and low cost. The Bluetooth specification defines a uniform structure for a wide range of devices to connect and communicate with each other.

Bluetooth enabled electronic devices connect and communicate wirelessly through short-range, ad hoc networks known as piconets. Each device can simultaneously communicate with up to seven other devices within a single piconet. Each device can also belong to several piconets simultaneously. Piconets can be established dynamically and automatically as Bluetooth enabled devices enter and leave radio proximity.

Spectrum
Bluetooth technology operates in the unlicensed industrial, scientific and medical (ISM) band at 2.4 to 2.485 GHz, using a spread spectrum, frequency hopping, full-duplex signal at a nominal rate of 1600 hops/sec. The 2.4 GHz ISM band is available and unlicensed in most countries.

Range
The operating range depends on the device class: class 3 radios have a range of up to 1 meter or 3 feet, class 2 radios, most commonly found in mobile devices, have a range of 10 meters or 30 feet and class 1 radios, used primarily in industrial use cases, have a range of 100 meters or 300 feet.

Data Rate
Bluetooth version 1.2 supports a bit rate of 1 megabit per second, while Bluetooth version 2.0 supports a bit rate of up to 2 megabit per second and Bluetooth version 2.0 plus EDR supports a bit rate of up to 3 megabit per second.

Bluetooth architecture
The Bluetooth specification describes the complete architecture from radio layer up to the service and device discovery protocol and applications. Figure 20 presents an overview of the Bluetooth stack. Bluetooth devices communicate with each other building and dynamically setting up the network. This is based on SDP (Service Discovery Protocol) that allows Bluetooth devices to discover services exposed by the others, and the establishment of a real communication channel between each pair of devices. The protocol that provides data exchanging services is the Logical Link Control and Adaptation Layer Protocol (L2CAP). It handles multiplexing and segmentation, through the use of the PSM (Protocol and Service Multiplexing) and SAR (Segmentation And Reassembly). Group abstractions and Quality of Service (QoS) features are supported as well. Higher-level protocols, such as RFCOMM or OBEX, are built over this basic layer.

RFCOMM provides emulation of serial connections; it supports framing and multiplexing and achieves all the required functions for serial data exchange. OBEX is built on top of RFCOMM to implement object exchange for objects like files and vCards.
Figure 20 - The Bluetooth stack

From a connection point of view, Bluetooth distinguishes two different links: ACL (Asynchronous Connectionless) and SCO (Synchronous Connection Oriented). The former supports packet-switched, point-to-multipoint connections and is typically used for data transmission over L2CAP. The latter supports symmetrical, circuit-switched and point-to-point connections and is typically used for 64kb/sec voice transmission, for which a minimum granted bandwidth is required. The radio and baseband layer, the two lower layers, are responsible for the physical wireless connections and controlling and sending data packets over this connection respectively.

**Device class**

Each device has its own unique address (e.g. 00:09:2D:56:61:F8) and publishes its Bluetooth friendly name, used to identify different devices to users, and the device class. The device class is used to define the type of device, which includes a major and minor device class and one or more service classes. The service class can vary from Positioning, Networking, Rendering, Capturing, Object Transfer, Audio, Telephony or Information, while the major device class indicates what kind of device it is: Computer, Phone, Network Access Point, Audio/video, Peripheral, Imaging, Wearable, Toy or Uncategorized. Each major device class includes a list of minor device classes. For example, the major device class Phone has the following minor device classes: Cellular, Cordless, Smartphone, Wired modem/Voice gateway, Common ISDN Access and Uncategorized. Using these device classes, a device is able to determine what kinds of devices are around it.

**Service Discovery**

Next to using the device class, SDP can be used to discover services exposed by other devices. A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A local device performing a service discovery on a remote device is able to browse all service records or search for a specific service. A service record contains among others the service class and service name. The service class defines the type of service and can be used by applications to determine whether a device supports a specific service.

**Connection establishment**

Two sequential phases are needed for establishing a communication channel (Anastasi et al., 2003). The first initial phase is called inquiry and it corresponds to device discovery from the master. The second phase is called paging and it corresponds to
initial connection setup. In both phases the master and the slaves perform different actions, which will be briefly summarized in the next sections.

**Inquiry phase**
A master begins device discovery by entering into the inquiry state and broadcasting messages in the following way. The inquiry hopping sequence is split into two 16-hop parts, called train A and B. A single slot lasts for 625s. On every even slot, the master sends two ID packets switching between two frequencies of the same train every 312.5s. On odd numbered slots, the master listens for slave answers. Since a single train will last for 16*312.5s and the slots for sending and listening are interleaved, two 10ms trains are defined. Each train must be repeated for at least $N_{\text{inquiry}} = 256$ times before a new train is used. In order to collect all responses in an error-free environment, at least three train switches must take place, so the inquiry state may have to last for 10.24s.

A slave that wants to be discovered enters in inquiry scan state. In this state, it listens for ID packets on the same 32 dedicated frequencies used from the master. The slave changes listening frequencies every 1.28s. The $T_{\text{w, inquiry scan}}$ time, during which the slave listens on the same frequency, must last enough to completely scan one train. $T_{\text{inquiry scan}}$ is the time between the beginning of two consecutive inquiry scan cycles and it will last at most 2.56 seconds. Both intervals can be changed. Default values are $T_{\text{inquiry scan}} = 1.28s$ and $T_{\text{w, inquiry scan}} = 11.25ms$.

**Page and connection phases**
Known the identity of the devices in its proximity, the master of a piconet explicitly pages them to join its piconet. First, the master selects the device to page and then it sends page messages on every frequency belonging to the two trains. Also in this phase, two intervals are defined to establish the amount of time in which the slave listens for page messages. Default values for page scan parameters are equal to inquiry scan default values: $T_{\text{page scan}} = 1.28s$ and $T_{\text{w, page scan}} = 11.25ms$

After several interactions, the master and the slave enter into a substate in which the clock input to the hop selection mechanism is frozen on the same value, and they can communicate using the same hopping sequence. At the end of the page, the master enters into the connection state and asks to open a connection. If the slave agrees, it acknowledges the request, the connection is finally established and the two devices can begin to exchange data on this connection.
C Interfaces

This section presents the interfaces between the nearcast client and nearcast center, and the interface between the nearcast center and nearcast server. These interfaces are defined by a list of procedures with a short description.

C.1 Interface between nearcast client and nearcast center

Table 33 - Procedure calls from nearcast client to nearcast center

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_nc_info()</td>
<td>Nearcast client requests information about the nearcast center</td>
</tr>
<tr>
<td>get_nearby_devices()</td>
<td>Nearcast client requests a list of nearby devices</td>
</tr>
<tr>
<td>location_subscribe(subscriber, subscription)</td>
<td>Nearcast client subscribes to location updates of a specific address</td>
</tr>
<tr>
<td>location_unsubscribe(subscriber, subscription)</td>
<td>Nearcast client unsubscribes to location updates of a specific address</td>
</tr>
<tr>
<td>download_content(filename)</td>
<td>Nearcast client requests to download the file with filename</td>
</tr>
<tr>
<td>get_downloadable_files()</td>
<td>Nearcast client requests a list of files that can be downloaded</td>
</tr>
<tr>
<td>new_image(filename, content)</td>
<td>Nearcast client uploads content</td>
</tr>
<tr>
<td>send_message(recipient, time, body)</td>
<td>Nearcast client sends a user message to the nearcast client that has to be delivered at a specific address</td>
</tr>
</tbody>
</table>

Table 34 - Procedure calls from nearcast center to nearcast client

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule_update(info)</td>
<td>Nearcast center sends a schedule update, information about the current and upcoming performances</td>
</tr>
<tr>
<td>nearby_devices_set(list)</td>
<td>Nearcast center returns a list of nearby devices</td>
</tr>
<tr>
<td>location_update(address, device_class, name, time, centerID)</td>
<td>Nearcast center updates the address, name, time and location of a specific device</td>
</tr>
<tr>
<td>return_content(filename, content)</td>
<td>Nearcast center returns the requested content</td>
</tr>
<tr>
<td>return_downloadable_files(list)</td>
<td>Nearcast center returns a list of downloadable files</td>
</tr>
<tr>
<td>send_message(from_addr, from_name, to_addr, time, message)</td>
<td>Nearcast center delivers a user message</td>
</tr>
</tbody>
</table>

C.2 Interface between nearcast server and nearcast center or external components

Table 35 - Procedure calls from nearcast center to nearcast server (procedures marked with * cannot be called by external systems)

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register(nc_name)</td>
<td>A nearcast center or external component registers itself at the nearcast server.</td>
</tr>
<tr>
<td>update_context(type, value)</td>
<td>A nearcast center or external component updates a piece of context information on the nearcast server.</td>
</tr>
<tr>
<td>update_location(address, device_class, name, time, )</td>
<td>A nearcast center or external component updates the nearcast server about the address, name, time and location of a device.</td>
</tr>
</tbody>
</table>
centerID) *

<table>
<thead>
<tr>
<th>Procedure call</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_connected(address, status) *</td>
<td>A nearcast center informs the nearcast server that a device is connected to the nearcast center.</td>
</tr>
<tr>
<td>send_message(from_addr, from_name, to_addr, datetime, body)</td>
<td>A nearcast center or external component sends a message, which the nearcast server has to pass to another nearcast center to deliver it at the target address.</td>
</tr>
<tr>
<td>location_subscribe(subscriber, subscription, status)</td>
<td>A nearcast center or external component passes a location subscription.</td>
</tr>
<tr>
<td>upload_content(from, location, content)</td>
<td>A nearcast center or external component passes uploaded content to the nearcast server.</td>
</tr>
<tr>
<td>get_downloadable_files(address)</td>
<td>A nearcast center or external component requests a list of downloadable content, to pass it to the nearcast client</td>
</tr>
<tr>
<td>download_content(filename)</td>
<td>A nearcast center or external component requests to download a piece of content.</td>
</tr>
</tbody>
</table>

Table 36 - Procedure calls from nearcast server to nearcast center
D Screenshots nearcast client application

This thesis’ prototype is developed for the music festival. Users walking around the festival terrain are able to do several things with the nearcast client application. This appendix presents a number of screenshots to give the reader an idea of what the application looks like.

Figure 21 - (a) Main menu of nearcast application. (b) Stage information: current and upcoming performance. (c) Stage details. (d) Buddy list showing location and time of all friends.

Figure 22 - (a) Find new buddies. (b) Composing a new message. (c) Choose a picture to download. (d) Download progress view.

Figure 23 - (a) Transfer complete. (b) Upload photo. (c) Choose file to upload. (d) Upload progress view.
E  Nearcast center design issues

This appendix presents some issues with the code design of the nearcast center. It is intended for programmers that are going to continue to develop the nearcast framework. It is assumed that the reader is familiar with the nearcast architecture.

**Blocking versus non-blocking sockets**

The nearcast center maintains exactly one connection with each client. This connection is used by both components: the nearcast center sends data to the nearcast client and the nearcast client sends data to the nearcast center. Thus, both components can start a ‘conversation’, but should also listen for incoming ‘conversations’. To be able to implement this, the nearcast framework uses non-blocking sockets.

Non-blocking sockets can be polled if there is any data ready to be read from the socket, i.e. an incoming conversation. If there is nothing to read, the application continues to check if there is any data ready to be written on the socket: the nearcast client then starts a conversation. Blocking sockets, the opposite type of sockets, cannot be used for such situation because the application blocks when reading from a socket, i.e. the read operation on a socket returns when there is data written on the socket. However, when the application itself needs to start a conversation, the socket is used by the read operation and thus cannot be used to send data to the remote device. This is the main reason that non-blocking sockets are used in the nearcast framework.

However, the problem with non-blocking sockets is that when the nearcast client sends an image of more than 100KB, some of the data is lost. The reason for this is that non-blocking sockets do not wait for the other side to be ready to receive more data. The sending side just pushes all data to the receiving side and does not check if the data has arrived. To overcome this limitation, the sending and receiving of data was modified such that the client waits for an acknowledgement of the receiving side each 8196 bytes. This causes a small delay in sending larger object, but guarantees a complete transfer.

Obviously, this is not a desirable situation. There are two different directions to think of possible solutions:

1. Actually, the OBEX profile should be used to transfer objects. However, the PyBlueZ API did not support OBEX connections.
2. Multiple connections between the nearcast client and nearcast center could be used to split the communication: one connection for sending data from the nearcast center to the nearcast client and one connection for sending data from the nearcast client to the nearcast center.

Future work should explore the feasibility of these solutions.

**Select versus threads**

The nearcast application has only one connection with the nearcast center. The nearcast center’s communication interface tries to read and write to the socket with the nearcast center. The nearcast center, however, can be connected with multiple devices resulting in multiple connections that have to be read or written. This can be handled by creating a thread for each nearcast client or using the `select`-statement.
Using threads is rather easy, because the same application logic from the nearcast client can be used. However, experiments showed that PyBlueZ raises Bluetooth errors (‘Error accessing Bluetooth device’) when different threads access the same socket. Apparently, passing the socket handler to another thread is not allowed.

To avoid these problems, the nearcast center uses the `select` statement, which allows polling a list of sockets. When one or more of the sockets have data ready to be read, `select` returns with a list of readable sockets. The function should also be able to write to sockets, but this becomes rather complex. For simplicity, another solution is chosen: the `select` statement returns after 0.2 seconds and then checks whether there is any data ready to be written.

Future work should include studying the following issues:

1. Find out why passing socket handlers to other threads results in Bluetooth errors. A possible solution for this situation could be to create a thread with a server socket that listens for incoming connections. When a new connection is accepted, the same thread is going to handle this connection. The server socket is destroyed and a new thread is started that creates a new server socket. In this way, sockets are only called within one thread.

2. Writing information should be handled by sending the message to a socket that is also put into the list of sockets to which the `select` statement is listening. When messages arrive on this socket, the `select` statement knows that there is data ready to be written to one of the connections. The `select` statement does not need to return after each 0.2 seconds anymore.

The second item should be studied first, because that would be the most elegant solution. When this issue is solved, the throughput of the connection between the nearcast center and nearcast client should be more optimal.