CCS – Collect, Convert and Send

– Designing and implementing a system for data portability and media migration to mobile devices

Fredrik Alserin
Jonas Gustafsson
Abstract

In this thesis we will identify which are the desired features and functionalities for implementing a system capable of acting as an information bridge for content available in the “wired” Internet to be delivered to mobile devices. We will also explore how to design and build such a system based on the specifications within parts of the MUSIS project. The MUSIS’ system development is used as a base of the work described in this thesis and the experiences from those efforts are used in order to design a system with more focus on data portability and media migration.

During the development of the MUSIS platform, problems related to system upgrading, i.e. adding new ad-hoc functionalities were discovered. Due to the fact that a user-centred design approach was taken, this was essential in the project. To solve some of these issues, we propose a new component-based system with a high level of scalability and re-usability. We name this system Collect, Convert and Send, CCS. The system shall be seen as a base that can be used as a core system for different projects where interoperability of content between different platforms, devices or systems is important.

The implementation of the system is based on the use cases and those theoretical aspects and ideas related to component software, interoperability, media migration and metadata in a Web service context. The results of our efforts give some indications that the use of component software gives a foundation for a service-oriented architecture.

Keywords: data portability, media migration, component software, metadata, interoperability, Web services, automatic content conversion, service oriented architecture SOA
Acknowledgements

We would to express our gratitude to the co-founder and head of the CeLeKT research team, Dr. Marcelo Milrad. We would also like to thank all the members of the CeLeKT team that have helped us in any way with our work.

The participants of the different projects that use the CCS system also deserve a special appreciation, especially the MUSIS and AMULETS projects.
Contents

1 INTRODUCTION ..............................................................................................................................6
  1.1 MOTIVATION ..............................................................................................................................6
  1.2 THESIS GOAL ............................................................................................................................6
  1.3 METHOD .....................................................................................................................................7
    1.3.1 Background: From MUSIS to CCS ....................................................................................7
    1.3.2 Methodological approach ...................................................................................................7
    1.3.3 Extending CCS to other domains .......................................................................................8
  1.4 TECHNICAL APPROACH ..........................................................................................................8
  1.5 EXPECTED RESULTS ................................................................................................................8
  1.6 LIMITATIONS ............................................................................................................................8
  1.7 STRUCTURE OF THE THESIS ...................................................................................................8

2 THEORY ..........................................................................................................................................10
  2.1 INTEROPERABILITY AND MEDIA MIGRATION USING SOFTWARE COMPONENTS AND METADATA 10
  2.2 COMPONENT SOFTWARE .......................................................................................................11
    2.2.1 Background ........................................................................................................................11
    2.2.2 Definition and benefits ........................................................................................................12
    2.2.3 Examples of existing component systems ...........................................................................13
  2.3 INTEROPERABILITY ................................................................................................................14
    2.3.1 Service Oriented Architecture (SOA) ................................................................................14
  2.4 METADATA ................................................................................................................................15
    2.4.1 Metadata introduction .........................................................................................................15
    2.4.2 Metadata generation ...........................................................................................................17
  2.5 STATE OF THE ART ................................................................................................................19
    2.5.1 Content Management Systems (CMS) .............................................................................19
    2.5.2 Postnuke ............................................................................................................................19
    2.5.3 Apache Lenya ....................................................................................................................20
    2.5.4 MMBase ............................................................................................................................20
    2.5.5 FirstClass ........................................................................................................................20
    2.5.6 CMS conclusions ...............................................................................................................20
  2.6 CHALLENGES FOR DESIGN ..................................................................................................21

3 RATIONALE AND DESIGN OF CCS ............................................................................................22
  3.1 REQUIREMENTS .......................................................................................................................22
    3.1.1 Functional requirements ......................................................................................................22
    3.1.2 Non-functional requirements ..............................................................................................23
  3.2 USE CASES ............................................................................................................................23
    3.2.1 Basic use case model of the CCS System ..........................................................................24
    3.2.2 Use case 1 and 2: Collect content from e-mail account ....................................................25
    3.2.3 Use case 3 and 4: Convert RSS feed to AMR and send to e-mail ......................................25
    3.2.4 Use case 5: Collect attachment from an email account and send the attachment to a
      FirstClass conference ...............................................................................................................25
    3.2.5 Use case 6: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via e-
      mail .........................................................................................................................................26
    3.2.6 Use case 7: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via the
      MUSIS news channel .............................................................................................................26
  3.3 ANALYSIS AND DESIGN .........................................................................................................26
    3.3.1 Design rationale ................................................................................................................26
    3.3.2 Technical approach ...........................................................................................................30

4 TECHNICAL IMPLEMENTATION, SCENARIOS AND ASSESSMENT ....................................33
  4.1 SOFTWARE AND SERVERS ....................................................................................................33
  4.2 SOFTWARE COMPONENTS ....................................................................................................34
    4.2.1 Web services used in the CCS system ...............................................................................35
  4.3 USE CASE ANALYSIS (UC4) USING UML SEQUENCE DIAGRAMS ....................................36
  4.4 HOW TO ADD A NEW CONVERSION METHOD WITHOUT CHANGING ANYTHING IN THE CLIENT SIDE 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCUSSION</td>
<td>43</td>
</tr>
<tr>
<td>5.1 Did we reach our goal?</td>
<td>43</td>
</tr>
<tr>
<td>5.2 What is our contribution to the field?</td>
<td>44</td>
</tr>
<tr>
<td>5.3 What should we have done differently?</td>
<td>44</td>
</tr>
<tr>
<td>5.4 Future development</td>
<td>45</td>
</tr>
<tr>
<td>5.5 Conclusions</td>
<td>45</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>46</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>48</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>49</td>
</tr>
<tr>
<td>LIST OF FOOTNOTES</td>
<td>50</td>
</tr>
<tr>
<td>APPENDIX A – USE CASES</td>
<td>51</td>
</tr>
<tr>
<td>APPENDIX B – SOFTWARE AND SERVERS IN THE CCS SYSTEM</td>
<td>57</td>
</tr>
<tr>
<td>APPENDIX C – JAVABEANS USED IN THE CCS SYSTEM</td>
<td>58</td>
</tr>
<tr>
<td>APPENDIX D – GUI AND JAVA APPLICATIONS USED IN THE CCS SYSTEM</td>
<td>59</td>
</tr>
<tr>
<td>APPENDIX E – DATABASE STRUCTURE USED IN THE CCS SYSTEM</td>
<td>60</td>
</tr>
<tr>
<td>APPENDIX F - SOURCE CODE FOR METHOD AIFF2AMR IN CCSCONVERTBEAN</td>
<td>64</td>
</tr>
</tbody>
</table>
1 Introduction

In this chapter we will give an introduction to this thesis. We start by explaining the motivation why this thesis was initiated in the first place and then the thesis goal is introduced. Further on we will discuss the methods used, the technical approach, the expected results and the limitations of the thesis. We end this chapter by showing the structure of the thesis.

1.1 Motivation

Recent developments in wireless technologies and cellular infrastructure allow end users to send and receive multimedia content to a variety of mobile devices, in particular mobile phones. These phones have the capability to store and play high quality multimedia content. Thus, these ubiquitous and inexpensive devices allow us to access and manipulate information almost anytime and anywhere.

On the one hand, this technical infrastructure opens new possibilities with regard to mobile devices usage and information access. On the other hand, there is a gap between the content to be delivered using this mobile network and the information and content that already exists in the “wired” Internet (Roman, Islam & Shoaib 2005). Mobile phone users are experiencing some limitations with regard to access and retrieval of information and sources available on today’s Internet. Thus, one of the problems that deserves further exploration deals with developing methods and techniques in order to reuse and convert this information available in the “wired” Internet into a suitable format that can be readable by mobile phones.

MUSIS (Multicasting Service and Information in Sweden) (MUSIS 2005) is a VINNOVA funded on-going research project that started in 2004. The main objectives of the MUSIS project are to explore, identify and develop a number of innovative multicast services with multimedia information to be distributed over wireless networks using multicasting solutions at university campuses (Milrad, Bergman & Jackson 2005). The authors of this thesis were involved in the MUSIS project as software developers and during the development of the MUSIS platform we discovered a number of problems related to system upgrading, i.e. adding new ad-hoc functionalities. Due to the fact that a user-centred design approach was taken, this was essential in the project. In one case the users wanted to have a new functionality, they wanted to have reminders from calendars sent to the phones. This proposal was raised during one of the workshop we had during the MUSIS project.

To solve some of these issues, we propose a new component-based system with a high level of scalability. We name this system Collect, Convert and Send, CCS. The system shall be seen as a base that can be used as a core system for different projects where interoperability of content between different platforms, devices or systems is important.

1.2 Thesis goal

The main issues to be explored in this thesis are twofold; first; we will identify which are the desired features and functionality for implementing a system capable of acting as an information bridge for content available in the “wired” Internet to be delivered to mobile devices and second; we will explore how to design and build such a system based on the desired specifications. Thus, the main two questions we will try to answer in this thesis can be defined as follows:

*How can content already available in the Internet be re-used and how can we design a system that supports data portability and media migration?*
We define media migration as the process of extracting media content from one context to another without changing the actual content.

These questions are too broad and generic to be completely solved within the scope of this thesis. Thus, we will elaborate on how to design such a system and thereafter we will illustrate these aspects by implementing parts of the CCS system in the specific context of the MUSIS project. The system requirements for the application we plan to design and implement are described in section 3.1.

1.3 Method

In this section we first describe the background in more detail, thereafter our approach based on that.

1.3.1 Background: From MUSIS to CCS

As mentioned in the motivation section (see section 1.1) we discovered some issues related to scalability and maintainability when designing and implementing a system more or less on-the-fly, i.e. create and/or upgrade an existing feature in a running system. This approach was taken because of the user-centred design that served as a base for discover and design of new features of mobile services. The users were able to be active participants of the design process by contributing through workshops and different questionnaires.

As any other project, time is an important factor and constraint. The development time of the MUSIS system was limited to when the students took the courses and the features had to be implemented and tested within that scope. As a consequence of this, features had to be implemented fast and efficient. In order to do so, the number of persons involved with the actual software development that built up the system was limited to as few people as possible. This made it hard for other people afterwards to “get into the code” and make changes if that person got sick or had to leave the office for some other reason.

Another problem was the rapid growth of new functionality. We discovered that when we added more functionality, the software structure of the system did not support the expanding complexity. It became too complex to be one large software component.

1.3.2 Methodological approach

In order to avoid the above-mentioned issues, some other approach needs to be taken. In this thesis, we suggest a component-based approach where features can be used and added in a more flexible way. The features of the original MUSIS system are closely connected with the MUSIS platform and do not enable other projects to use its functionality. To use the MUSIS features in other projects, we have to open up some services by providing a component interface into the new CCS system.

We suggest a design approach were we first define the requirements of the system (see section 3.1). After that, we describe the system using a use case model accompanied with more detailed use case descriptions (see section 3.2). We will implement the system based on the use cases and the theoretical foundation in the following chapter (see chapter 2).

As the system development is a part of a VINNOVA-MATIMOP funded project with deadlines and continuously growing number of desired features, the system has been developed using evolutionary path design perspective as well using the spiral model for iterating through ideas and new demands directly from the users. This user-centered approach is now widely accepted and also the iteration point of view. The “design, test, measure, and redesign” should be repeated as often as possible (Preece, Rogers & Sharp 2002), and with user involvement. This does not mean that it is always
a good idea for users to be designers, but as long as the designers remain aware of the users while making their decisions, the principle can be upheld.

In the design chapter, we will use standard UML use cases and use case scenarios for describing the flow of events and the requirements that are specified. Use cases are defined by (Rumbaugh, Jacobson & Booch 1998) as:

“A specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem or class can perform by interacting with outside actors” (Rumbaugh, Jacobson & Booch 1998)

In other words, by using this method we could describe how the system will behave in a standardized way. Use cases is good to use when you have a system with lot of interfaces (actors or components) and when there are mostly functional requirements (Arlow & Neustadt 2002).

As these two criteria will be fulfilled in the CCS system this is a method we will utilize in this thesis.

1.3.3 Extending CCS to other domains
As mentioned in the background section 1.3.1 one issue with the MUSIS system was the growing complexity and difficulty of adding new components. The improved CCS system should tackle these issues and be prepared for extending itself to other domains. For example, it should be able to use only the repository feature of the CCS in on domain, and only the collecting mechanism in another. In other words, the system should be to act as different types of middleware, or as an independent system.

1.4 Technical approach
To implement the system we are proposing, we plan to use technologies that support a component based architecture. To achieve this we will use the Java programming language and Web services. This is described in section 3.3.2.

1.5 Expected results
As a result of our efforts, we expect to develop a functional prototype of the CCS system, containing the specifications that will be described in section 3.1.

1.6 Limitations
The thesis will not focus on how to produce content that are suitable for mobile devices. Furthermore, we will not investigate which formats that are suitable for conversion to mobile devices neither content distribution techniques using wireless multicasting solutions.

Since the functionality has been tested and evaluated within the MUSIS project (Milrad, Bergman & Jackson 2005), we will not emphasize on that within this thesis. We will not test the CCS system with respect to performance and administration of the system. The graphical user interface (GUI) that we will develop in this thesis shall be regarded as a tool for demonstrating the functionalities in the CCS system. The GUI we design is not meant to be included when extending CCS to other domains.

We will not focus on how to provide complete user management system when it comes to sending the content that is available in the CCS system.

1.7 Structure of the thesis
The thesis is divided into five chapters. The first chapter presents our initial ideas and the methodology used. In chapter two, we then go into details regarding the theoretical foundations in order to give the reader the knowledge needed to better understand the
problem under investigation. The coming chapter starts by presenting the requirements of the system. We then analyse the requirements with use cases and then present a design based on this. In chapter four we present the implementation of the suggested design from the previous chapter. Finally, in chapter five, we discuss the thesis outcome and conclusions.

In the technical description of the system, we decided to describe our implementation by highlighting different parts of the system in a generic technical manner rather than showing actual code and describe it row-by-row. Some sample code of the most important parts is available as an appendix for the especially interested reader.
2 Theory

The aim of this chapter is to give the reader the foundations needed in order to understand some of the core concepts discussed in this thesis. We will introduce the terms software component, interoperability and metadata and we will show in section 2.1 how these concepts are integrated. In section 2.2 we will elaborate on the term component software and in section 2.3 we extend this to interoperability. Important notions about metadata are described in section 2.4, and finally in section 2.5, we look into some state-of-the-art examples in this particular field.

2.1 Interoperability and media migration using software components and metadata

The goal of this thesis is to design a system that is capable of transforming content from one context to another. To be able to do this, the system and its underlying components must have specific knowledge about the content and the device that the content is sent to. The system we will design is not in charge of creating content but rather how to collect already created content, transform it and finally send it to different devices. In order to perform these tasks the system need to have some special features. These features, or characteristics, can be seen as interplay between software components, the interoperability of the content between platforms and systems and metadata that interoperability can be relied upon.

As illustrated in Figure 2-1, there must be interplay between the three parts of the model. If an object is to be sent to a device, the system must know something about the object. For example, if a picture is to be sent from a repository of objects to a user, the system may take the following attributes into account before sending it:

- What formats can the receiver handle?
- When is a suitable time for sending the content?
- How does the receiver want to get the image?
- How large content want or can the receiver handle?

We see here that data about the content and the receiver is of most importance. The system must be able to collect some data concerning the object and then store it, or at least be able to get the information at runtime. This data about the data is commonly known as metadata and will be described in more detail in section 2.4. Figure 2-2 illustrates that it is not possible to send content in formats that the mobile phone cannot handle and that the need for a middleware in that case is obvious.
As seen in Figure 2-2, the intention is to send content by using some middleware software that supports the interoperability of the content between platforms and devices. This process can be referred to as media migration. In the scope of this thesis, we can see this ability of the system as a function to make it possible for one device to consume content that it was not intended for.

In the next section we will describe the fundamentals on component software and how it relates to our work. We will then in section 2.3 describe interoperability and in section 2.4 the term metadata.

2.2 Component software
One of the features to be investigated in this thesis is how to design a flexible architecture with scalability as a high priority; component software is a way of achieving this. In this section we will talk about component software, what it is and what its benefits are.

2.2.1 Background
As early as in 1968 the expression “Software crises” was invented at NATO Conferences on Software in Garmisch, Germany. According to Cox (1992) this was

“to indicate that software was already scarce, expensive, of insufficient quality, hard to schedule, and nearly impossible to manage”. (Cox 1992)

In 1987 Brooks wrote the classic paper “No Silver Bullet” where he claims that there will be no more silver bullets i.e. there will not be any solutions to solve the difficulties of software engineering (Frederick P. Brooks 1987). He uses the metaphor of a werewolf as a software project, and like a werewolf a software project

“[…] has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet--something to make software costs drop as rapidly as computer hardware costs do[…]” (Frederick P. Brooks 1987)
The central argument that Brooks uses is the distinction between accidental and essential complexity. Accidental complexity is something that we make when we are trying to solve a problem and can be fixed but essential complexity is caused by the problem to be solved and cannot be fixed. In Brooks’ opinion, software engineers have solved most of the problems with accidental complexity, for example, the invention of high-level languages have improved software productivity, reliability, and simplicity. The essential complexity is in the nature of software according to Brooks and cannot be reduced. Cox holds a totally opposite view of this issue, in his opinion, there are a silver bullet. But it is not a new technology that will slay this monster.

“The silver bullet is a cultural change rather than a technological change. It is a paradigm shift; a software industrial revolution that will change the software universe as the industrial revolution changed manufacturing.” 
(Cox 1992)

This paradigm shift is according to Cox how we buy, sell, and own software. We cannot use the same ideas as we do when we are talking about buying, selling, and owning tangible goods; software does not consist of atoms instead it consists of bits.

“[…] buying, selling and owning is based on acquisition of utility encoded in the bits instead of acquisition of the bits themselves […]” (Cox 1992)

By using components in software development, we can make services of these components and thus sell the service the component provides instead of the component itself. According to Cox’ belief, we will slay the werewolf or at least make an attempt to do so.

In our thesis, we will use this approach to achieve a new component-based system with a high level of scalability. So what is a component? This we will try to describe in the following section.

2.2.2 Definition and benefits

There are a lot of definitions of what a component is. Szyperski’s definition is often used:

“A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.” 
(Szyperski 2004)

Völter examines this definition and he says that Szyperski state that a component’s purpose is to be composed with other components.

“A component-based application is thus assembled from a set of collaborating components.” (Völter 2003)

To be able to assemble components into systems or applications, every part needs to have a clear interface. The services a component provides shall be stated by these interfaces and doing so the interface defines the components responsibility. If a component depends on another component, this dependency must be explicitly specified.

A component is self-contained; if you change anything inside a component, you do not need to change anything in other components as long as you do not change the interfaces. Völter claims that it’s not necessary that it’s the same people how make the components that later one will assemble these components to an application. (Völter 2003)
By using components in software development, we can make the job faster and more reliable.

“Less new code must be written to produce the same results. Off-the-shelf components should be “well-seasoned” and therefore more reliable than code written from scratch.” (Kaisler 2005)

Using (Lau 2004), Szyperski identifies the following three sets of arguments in favour of component software (Szyperski 2000):

- Baseline argument: Strategic components are made while non-strategic ones are bought.
- Enterprise argument: A core set of components can be configured in the products of one product line in different ways. In addition, product specific components may also be configured in the products. Product evolution can be controlled by versioning components and reconfiguring systems.
- Dynamic computing argument: Modern open systems can be dynamically extended.

In the section we will list some existing component systems that demonstrate the definitions and benefits in this section.

**2.2.3 Examples of existing component systems**

There is several existing component system, we will in this section briefly list the most common ones:

- **CORBA (Common Object Request Broker Architecture)**
  - CORBA\(^1\) is OMG’s (Object Management Group) architecture for applications when they want to interact with other applications over networks. It is open, vendor-independent and can be used on any operating system, programming language and network.

- **COM (DCOM Component Object Model Technologies)**
  - COM\(^2\) is developed by Microsoft and it enables software components to communicate. It is used to create re-usable software components, link components and to take advantage of Windows services. Subsets of COM is; COM+, DCOM and ActiveX.

- **EJB (Enterprise JavaBeans)**
  - EJB\(^3\) is a server-side component architecture for Java developed by Sun which can communicate with other EJB via Remote Method Invocation (RMI). EJB is used on the J2EE (Java 2 Enterprise Edition) platform and makes it easier and faster to develop transactional, distributed, secure and portable applications.

- **Web services**
  - Web services are based on XML-standards\(^4\) such as SOAP\(^5\), UDDI\(^6\) and WSDL\(^7\). It is the best implementation of SOA (Service Oriented Architecture) so far.

  “Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.”\(^8\)
• .NET
  o .NET\textsuperscript{9} is a framework for Web services developed by Microsoft. .NET enables developers to create XML Web services and integrate mobile devices and servers.

All of these examples are systems that are used today, but all of them except Web services have major problems with interoperability. We will in the following chapter explain what interoperability is.

2.3 Interoperability
The term interoperability comes from inter, which means between, and operate. This means that information and data should be able to operate and to be shared by different applications, system or devices, depending on the situation. In this thesis, the interoperability issue is of great importance since the produced and aggregated content should be used in different environments with different capabilities. For example, the daily lunch menu on a website should be transferred and viewed on a mobile device within a web browser. In most of the cases, we need some conversion process to take care of this since the level of interoperability between the information on the wired Internet and a mobile phone or smart phone is poor. Recent efforts, for example Opera Mini, try to make a bridge between “normal” web pages and automatically adapt them for small devices. In that case, the data to be shared between different environments are limited to data on the web.

There are other types of data that needs to be able to operate between systems, for example messages from an educational platform within a university course, a schedule in the iCalendar format etc. These types of content are not easily transferred to a mobile device and some conversion process other than Opera Mini is needed. For a good user experience, the conversation process should be transparent to the user. (Riekki, Salminen & Alakarppa 2006) uses Gaia (Roman, Islam & Shoaib 2005), one.world (Grimm 2004) and Aura (Garlan et al. 2002) to identify the fundamental requirements for a generic-use pervasive middleware for mobile devices:

- **Interoperability**: the ability of two or more systems or components to exchange information and to use the exchanged information;
- **Discoverability**: the ability for system components to discover surrounding entities and, conversely, for other entities in the surroundings to discover the system components;
- **Location transparency**: the ability for system component locations to be transparent to other components, the programmer, and the user;
- **Adaptability**: the ability of a software entity to adapt to a changing environment;
- **Context-awareness**: the ability to have awareness of the user’s context, which enables a pervasive system to provide relevant information and services to users.

This leads up to a brief overview of service-oriented architecture (SOA).

2.3.1 Service Oriented Architecture (SOA)
Recent efforts show that services are becomes more important rather that using one product that takes care of all things. Although not a new phenomenon, SOA and API (Application Programming Interface) programming is becomes more and more popular, especially when considering Web 2.0 web sites and their underlying architecture. More traditional SOA include DCOM and CORBA while more recent efforts tries to use web
services as a method for connecting and transfer data between different platforms and systems, thus enabling interoperability. There are no standards defined by W3C of what a Web 2.0 web site may look like, but it is clear that Web 2.0 can be regarded as displaying some SOA characteristics. Web 2.0 can even be said to be the most massive instance possible of service-oriented architecture (Hinchcliffe 2006).

Web 2.0 web sites often uses web services or other XML-based approaches when it comes to requesting a service from a service provide. Just to mention a few, services provided by different organizations could include;

- Tagged images from Flickr
- Latest news from CNN
- Latest radio show from Swedish national radio (Sveriges Radio)
- Search results from Google

In these four cases, we could request the data using many types of formats. We could request the images from Flickr by using RSS or a Web service call; the news from CNN could be fetched using RSS and the radio news from Swedish national radio using RSS 2.0 with enclosures. The Google search could be requested using SOAP, for instance.

The basic philosophy is simple; some service provider announces their service and others are free to use them. But SOA is more than just aggregating a news feed, it’s also about how the service itself is organized and built up, the modularity and interoperability, the architecture behind it (Balzer 2004).

By using SOA and services that support SOA in our system we can achieve an open architecture that can help when we want to add new functionalities to our system.

2.4 Metadata

As seen in section 2.1 we need some way of describing the data that we collect. Metadata is a solution for this and in this section we will give a brief introduction to the field of research. We end the section by describing current efforts in the field of mobile metadata generation.

2.4.1 Metadata introduction

Metadata is data that describes data. The metadata can be when a photo was taken, who took it and where. It can also be more complicated metadata, such as why it was taken and what it is. You can also derive the “when” attribute in many different ways, for example:

- 2006-06-07
- 1254865750
- In the summertime
- Early morning
- Day after dad’s birthday

As we can see above different time stamps mean different things to different people in different types of situations. The semantics is very important to the user, but it’s hard for a computer to associate a precise time with “Day after dad’s birthday”. The need for a standard on how to describe a digital object to satisfy both human and computer needs has been up for discussion for several years. As stated before, the need for a standard is
different depending on the situation, and therefore many types of standards have been discussed. (Stamou et al. 2006) identifies six important standardization activities:

- **Dublin Core Community**, provides a small set of descriptors that quickly drew global interest from various information providers in the arts, sciences, education, business, and government sectors.
- **MPEG-7**, a standard for describing the multimedia content data that supports some degree of interpretation of the information meaning, which can be passed onto, or accessed by, a device or a computer code.
- **“MPEG-21 Overview v.5** aims at defining a normative open framework for multimedia delivery and consumption for use by all the players in the delivery and consumption chain.
- **NewsML**, designed to provide a media-independent, structural framework for multimedia news.
- **TV-Anytime Forum**, aims to develop specifications to enable audio–visual and other services based on mass-market high volume digital storage in consumer platforms—simply referred to as local storage.
- **Virtual Resource Association, VRA Core 3.0**, consists of a single element set that can be applied as many times as necessary to create records to describe works of visual culture and the images that documents them.

In order to enable intelligent and personalized processing, management and use of multimedia content and services, the use of application specific metadata descriptions and adequately formulated user profiles is most important. Concerning analysis of user image searching, (Fukumoto 2006) identifies three types of image databases:

- **Feature type (Flickner et al. 1995)** is based on the colors or shapes of object in the images. When the retriever puts in color or shape, the system starts searching directly the database with color histograms or shape.
- **Sensitive type (Kimoto 1999)**. The user inputs sensitive words and the system exchanges them for color information and searches the database.
- **Metadata type (Kiyoki, Kaneko & Kitagawa 1996)**. Each image in the database is given a metadata tag which defines its characteristics through text or digits. In order to search the database the user has to use the corresponding metadata tag.

There are now accepted ISO (International Standards Organization) regulations governing the structure contents description of metadata for multimedia known as MPEG-7. The structure and the framework of metadata need another consideration and this is considered and standardized in MPEG7. You need both objective keywords (e.g. “pond” or “monument”) and subjective keywords (“summer” or “shine”) when creating metadata for images (Fukumoto 2000).

The Dublin Core is a metadata standard for describing digital objects (including web pages) to enhance visibility, accessibility and interoperability, often encoded in XML. It was so named because the first meeting of metadata and web specialists, which saw its birth, was held in the town of Dublin, Ohio in the United States. The standard element set consists of 15 elements which may occur zero or many times in whatever order. Some of them are described here:

- **Title**, e.g. "A Pilot's Guide to Aircraft Insurance"
- **Creator**, e.g. "Internal Revenue Service. Customer Complaints Unit"
• Subject, e.g. "Aircraft leasing and renting"
• Description, e.g. "Illustrated guide to airport markings and lighting signals, with particular reference to SMGCS (Surface Movement Guidance and Control System) for airports with low visibility conditions."
• Publisher, e.g. "University of South Where"
• Date, e.g. "1998-02-16"
• Type, e.g. "Image"
• Format, e.g. "40 x 512 pixels"
• Identifier, e.g. "ISBN:0385424728"
• Source, e.g. "Image from page 54 of the 1922 edition of Romeo and Juliet"
• Language, e.g. "en-US"
• Coverage, e.g. "1995-1996"
• Rights, e.g. "http://cs-tr.cs.cornell.edu/Dienst/Repository/2.0/Terms"

The examples are taken from the official Dublin Core usage guide (Hillmann 2005). When these metadata names are going to be described, there is a chance that some other standard uses the same element or attribute name. This is where namespaces come in. A namespace is a collection of names that is identified by a URI reference (Powell & Wagner 2001). This definition is also used by W3C, and by assigning expanded names to element and attribute names the W3C namespace standard mechanism can cope with this issue (W3C 2006). Since namespaces are a common way of describing metadata and handle element and attribute collision, the system suggested in this thesis need to handle this.

2.4.2 Metadata generation
One of the issues in this thesis is how we can design a system that supports media migration. To be able to do that, for example collect a live video feed from the Internet and send it to a Nokia 6630 smart phone, we have know certain things about the original content and what types of content the receiver may handle. To support this, we must analyze the video feed from the streaming server and then store in a format that can be converted into video feed that can be sent to the phone. To be able to make the correct conversion, we must know enough data about the feed and enough facts about the receiver. With other words, we must have a method for collecting quality metadata. Depending on how we plan to use the content, the question of what quality metadata is varies.

One area where quality metadata generation is a hot topic is mobile phones with camera functionality. One issue is how to collect the metadata from the user. When the user takes a picture with the camera on their camera-enabled phone, we can let him/her input some keywords related to the picture directly after the picture is taken. We can also try to automate this process, avoiding cumbersome interactions for the user. The process of knowing what to gather without forcing the user to enter it can be referred to as a context aware system. The system can draw some conclusion upon certain basic things, such as what the user have done before, what time is it, where is the user, why is the user at that location, etc. More or less advanced approaches can be used to determine metadata about an object without asking the user at creation time. Some metadata can be determined afterwards and some can be synchronized with action or an event that already is known, for instance the schedule of the user. If we know that the
user is going to be on vacation in July and the picture is taken 3rd of July, we can guess the location based on that.

The easiness of finding an object in a database depends on what type of metadata we know and how that metadata is stored. On common approach on the web today is letting the user define keywords associated with an object, for example:

- Tagging of weblog entries
- Tagging of bookmarks
- Tagging images in Flickr

There are many things to take into account then in comes to what word to use when tagging a picture. Just to name a few, the user may speak different languages; s/he may spell the word wrong, leave out spaces or replace them with underscores etc.

One application that tries to solve some of the issues raised above, especially concerning automatic metadata generation, is Merkytis (Merkitys 2006). Merkytis is a way of letting the users automatically create the necessary metadata and then attach the data to a picture that you can take with the application. The application is developed for smart phones and can automatically collect some contextual metadata of your surroundings, such as your location, GSM (Global System for Mobile communication) cell info, country etc. The contextual information and the image can optionally be sent to an online photo and sharing management tool such as Flickr21. The following pieces of contextual information are sent via the application, if it can be collected:

- Location information
  - Global Positioning System, GPS, (with additional Bluetooth GPS device)
  - GSM Cell info
  - Country
  - City
- The Bluetooth environment
- Calendar events
- User defined tags
- User defined description

The feature that “stands out” is the way the application automates the process. Since the username and password to the Flickr account is stored in the phone and prepared in advance, there is no need for the user to enter it manually each time. If you want to, you can manually add or remove the tag words that the application builds up for you.

Using Merkytis is not limited to Flickr. You can set up your own server and make the application upload the pictures there instead. It can, however not fully implemented, also do audio and video captures and uploads.

The Merkytis application is similar to the work conducted at Garage Cinema Research at UC Berkeley's School of Information22. They have developed a similar application, Mobile Media Metadata 2 (MMM2), which also collects metadata and then attaches it to an image that can be shared to other users via the Internet. At capture time, MMM2 automatically records the following (Davis et al. 2005):
2.5 State of the art

In this section we will highlight state of the art projects and ongoing efforts in the field of CMS (Content Management Systems) and metadata generation. We start by briefly describing what a CMS is and from there we have chosen four different CMS that we will describe in more detail; Postnuke, FirstClass, Apache Lenya and MMBase. We conclude the chapter by showing a comparison table of the described system and relating them to the features that are important for this thesis.

2.5.1 Content Management Systems (CMS)

According to the SMPTE/EBU (Society of Motion Pictures and Television Engineers, European Broadcasting Unit) 1998 taskforce, the media content consists of essence and metadata (Mauthe & Thomas 2004). The essence is the material itself and metadata is the data that describes the essence. In order to be called a Content Management System, the system must handle both the essence and metadata that describes it.

A content management system for creating and managing web content can mean a lot of things these days. The most common belief of its definition and purpose is most likely that it is a web-based system that allows administrators to create and manage their web content. The web content can be standard web pages built using a WYSIWYG-editor (What You See Is What You Get) on the web but it can also be other types of content, for example images, movies, documents, polls or discussion forums.

In this thesis, we are interested in how the CMS’s handle digital assets, for example in form of rich media content, including images, text, videos, sounds etc. In more detail, we are interested in how the CMS can support interoperability for the content between different platforms and devices. To support that, we need the CMS to support the interplay between the components that build up the system and the metadata about the content (see Figure 2-1)

2.5.2 Postnuke

Postnuke is an open source content management system project that tries to be flexible and easily upgradeable by using modules (PostNuke 2006). It is based on PHP and MySQL. The functionality is a basic web site management system with extra gadgets. Examples of these extra things is that you can add a forum, a gallery etc. You have an administrator panel were you can mange the site via a web browser.

Considering the things we want to achieve with this thesis we have investigated in how Postnuke deals with storing content and the contents metadata. There is a download repository built into Postnuke but it does just have some basic metadata capability. You can via a third party community download modules that will expand the functionality of Postnuke. We found two modules, which are content management modules. The first, PagEd, has some metadata generation. When you upload a file you may add a description of the file, the other metadata that this module stores about a file is file name, size, topic, date and which user has uploaded the file. Mediashare is the name of the other module. You can upload media files into albums and share these files with other users with this module. With this module you are only able to upload these media formats; gif, png, jpg, flash, Quicktime, Windows media player and RealPlayer.
When you upload the content, you have the possibility to describe it using keywords, also called tags. After you have done that, you have the possibility to search the database for media with these keywords.

In our opinion this CMS is not powerful enough for our needs, regarding the metadata generation and storing. You have the possibility to develop your own module for Postnuke that will provide you with the functionality you wish for.

2.5.3 Apache Lenya

Apache Lenya is an open-source CMS based on XML and Java. It includes revision control, site management, page activation/deactivation scheduling, page search, WYSIWYG editors, and workflow (Lenya 2006). Apache Lenya is a powerful CMS, built upon Apache Cocoon. Due to this fact you can develop extra functionality that does not exist with Cocoon components.

Every page you create can have Dublin Core metadata assigned. If you have the need for another metadata standard this can be added. Different modules are shipped with Lenya; one is Lucene, which gives search functionality. But the most interesting is the JCR repository, where you can store content in repository. JCR stands for Content Repository API for Java Technology. The JCR repository is the same that is built in Cocoon, and therefore works in the same way. But the JCR in Lenya is a beta version and not fully implemented. When it is working properly the JCR will not have any limits in terms of the type of data it can hold. You can store a lot of different content such as images, videos and text documents.

2.5.4 MMBase

MMBase is a Web Content Management System with strong multi media features and advanced portal functionalities (MMBase 2006). MMBase has a large installed base in the Netherlands, and is used by major Dutch broadcasters, publishers, educational institutes, national and local governments. MMBase is written in Java, it is Open Source Software (MPL) and all standards used are as 'open' as possible. The system can be used with all major operating systems, application servers and databases.

MMBase highlights two important features that are within the scope of this thesis. It separates content from lay-out and structure. This means, for example, that a piece of text can be used on different sites, mobile phones, flash movies, print, CD, in a different context and lay out. It is also platform-independent when it comes to operating system, database and application server. However, MMBase will in the future move towards the J2EE platform.

2.5.5 FirstClass

FirstClass can be seen as a CMS with a long history, mostly within the learning industry. FirstClass has an integrated user management system and the current version also supports automatic generation of RSS feeds, web blogging, a plug-in mechanism and different templates for outputting content (FirstClass 2006).

The system also has an inbound and outbound API that can be used to send or get content from FirstClass from an outside system.

You can also script and automate events within FirstClass to trigger action from different when some event occurs. One example could be to generate an auto response if the headline contains a specific word or is from a specific user.

2.5.6 CMS conclusions

To make a comparison more readable and easy to comprehend, a table of the most important features for this thesis can be suitable. We have not tested the CMS systems I
real life, due to time constraints. The facts and the table below is a result of reading reviews and comparing the facts on the different projects web sites. Information in large matrixes over CMS should always be seen with sceptic eyes (cmsmatrix.org 2006), (HartmanCommunicatieBV 2006). To get a fully correct picture, all CMS’s should be tested and evaluated in more detail.

<table>
<thead>
<tr>
<th></th>
<th>Apache Lenya</th>
<th>PostNuke</th>
<th>MMBase</th>
<th>FirstClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Storing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Metadata management</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Automatic metadata generation</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Plug-in mechanism</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>High accessibility to content</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Schedule content storing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Content conversion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service oriented architecture (SOA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1 - Content Management System Comparison

As seen in table 2-1, most of the systems have many features and is capable of many different jobs. However, the table does not say anything about how well (if at all) the different features can operate between each other, which are an important factor. For example, it is relevant for this thesis if the newly installed plug-ins can be scheduled. It is also interesting to know if which types of plug-ins that can be inserted and in which parts of the system they can operate, i.e. in the content uploading, conversion, sending, metadata generation etc. Another interesting issue to highlight is that none of the above have a clear service oriented architecture, e.g. with a web service API to interact with other systems.

We can also see that no system is capable of all the features we highlight in this thesis. For example, Postnuke is not powerful enough for our needs, regarding the metadata generation and storing. You have the possibility to develop your own module for Postnuke that will provide you with the functionality you wish for. MMbase is a widely used and indeed advanced system with many features, but it does fulfill our needs regarding automatic content conversion. This is also true for Apache Lenya.

2.6 Challenges for design

In this section we have introduced several different concepts and challenges, such as interoperability, media migration, software components, metadata and some state or the art applications within the field.

In the introduction, we ask the question on how content already available in the Internet be re-used and how can we design a system that supports data portability and media migration. The question is indeed very broad and generic. To be able to support different kinds of content types and services on from the Internet available today, any user can pick one of the systems presented in the state of the art-chapter. The most challenging aspect though might be the design of a system that can be prepared for future content types and future Internet services to consume. Therefore, an easily maintained and expandable collect mechanism as well as an equally generic delivery mechanism is of most importance.
3 Rationale and design of CCS

As mentioned in section 1.2 the goal of the thesis is about to identify and define features and functionalities of a system capable of acting as an information bridge for content available in the “wired” Internet to be delivered to mobile devices. We will also explore how to design and build such a system.

These two issues are too broad and generic to be completely solved within the scope of this thesis. To be able to design and implement a prototype of such a suggested system, we will narrow it down to a specific context. Having this in mind and the discussion in section 1.6, we will design the CCS system based on the requirements and functionalities of the MUSIS system. We will also design the system with regards to the issues related to data portability and media migration.

As described in the method section 1.3, we will start by defining the requirements and then go into detail by using a use case model and its descriptions. We will then end this chapter by elaborate on the design rationale.

3.1 Requirements

Requirements can be divided into functional and non-functional requirements. A statement of what a system should do is called a functional requirement. On the other hand, a constraint or a specific property in a system is called a non-functional requirement (Arlow & Neustadt 2002).

In the ISO 9126-1 standard, Software Product Evaluation (ISO/IEC-9126-1:2001 2006), six categories is presented. They can be divided into two groups where the functionality aspect is a functional requirement and the rest can be seen as non-functional requirements for evaluating the quality of the product. The six aspects are:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

In our thesis, some are more important than others. In sections 3.1.1 and 3.1.2, we go further into this by specifying the functional and non-functional requirements.

3.1.1 Functional requirements

As about the particular functional requirements, the prototype of the system we are planning to implement shall be able to perform the following tasks:

- Collect different multimedia objects available on the Internet and convert them to the following content types as describe below:
  - MP3 -> AMR (audio)
  - RM -> AMR (audio)
  - RSS -> AMR (text-to-speech)
  - RSS -> HTML (text)
  - RTSP -> 3GPP (video and audio)
• It should be possible to add new conversion methods for existing and new formats.
• Collect mail from an e-mail server and then send the attached content
• Enable external user managements system (UMS) to be connected to the delivery mechanism of CCS
• E-mail, e.g. to a group e-mail account defined by an external UMS
• Send to a MUSIS content channel using the MUSIS UMS
• Podcasting of a user selected video, audio and image content using RSS with enclosures
• By means of a scheduling mechanism, the system shall automatically perform the above mentioned actions

3.1.2 Non-functional requirements
In the ISO 9126-1 standard, Software Product Evaluation (ISO/IEC-9126-1:2001 2006) there are five non-functional requirements. We will list them and show how they fit to the CCS system.

- Reliability - the system should not crash if the user does something wrong, instead the user shall be informed about the errors. This shall be done by implementing error handling into the system. The system shall also be able to be updated with new functionalities while the project is running.

- Usability - the system shall be easy to use, i.e. easy to interact with the functions that build up the system. The user shall be responsible for how their applications/systems user interface interacts with the CCS system. The CCS system is responsible for the interface between itself and the applications and/or systems that wants to interact with the functionalities of the CCS system

- Efficiency – the system shall perform its tasks as efficient as possible, by this we mean that we shall try to avoid redundancy in the system. The CCS system is not responsible for how external applications/systems perform, but this aspect should be taken in consideration when choosing which applications/systems shall be used.

- Maintainability – it is most important that is possible to upgrade and/or expand the system with new and/or improved functionalities. When an improvement of an existing functionality occur the user of the system (the client) should not have to change anything. The interface between the CCS system and the users shall not change. If a new functionality is implemented into the CCS system the user needs to add this functionality’s interface into the user’s application/system.

- Portability – it should be possible to move the CCS system to different servers, it should not be an issue on which platform the server is running i.e. which operating system the server is using.

3.2 Use cases
In the previous section, we saw that the requirements of the system demand a complex system with many components and mechanisms. In this section, we try to illustrate the design of the system by first showing a generic use case model and then go into detail with more specific ones. Each use case starts with a narrative form and then a more
formal use case description is presented in Appendix A. The following use cases will be examined:

- Collect content from e-mail account
- Convert RSS feed (text) to AMR (audio) and send to e-mail
- Collect attachment from an e-mail account and send the attachment to a FirstClass conference
- Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via e-mail
- Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via the MUSIS news channel

These use cases are selected so that the essential parts of the system can be illustrated. It also shows the variety of the functionalities that the CCS system provides.

### 3.2.1 Basic use case model of the CCS System

When considering the functional requirements, we can see that some sort of collecting mechanism is necessary. The collected content should be able to be converted and thereafter sent to some receiver. However, it’s not always the case that the content needs to be collected, it could already be in the system, for example a video file that needs to be converted into another format. In that case, it’s good if the video file already is in the system. A similar case is when a file should be sent to several different receivers. We do not want to collect and convert a file one time for each receiver. In other words, a user may want to collect content at one time, and convert it some other time. This simple description of the system can be seen in the following basic use case model:

![Figure 3-1 Basic use case model](image)

In Figure 3-1 we describe two different actors, one User and one Automatic scheduler. The actor User can manually collect and convert content or view the content either by downloading or sending the content. The Automatic scheduler can also collect...
and convert content as well as sending it. Please note that it makes no sense to enable DownloadContent for the Automatic scheduler since it should be seen as an end user action. To get content for conversion, the Collect action is used.

As seen in Figure 3-1, there is not a forced path in the system. The scheduler can either do all actions in a row or only one. For example, one action could be to collect an RSS feed, convert the text of an item and then send an audio version of that text as an e-mail attachment. Another action in the system could be to simply collect e-mail attachments and store them for later use. The following section describes these types of use cases in more detail.

3.2.2 Use case 1 and 2: Collect content from e-mail account

One functional requirement is the possibility to send content that has been stored in the database via e-mail. For example, the user may have taken a picture with the camera in a mobile phone and wants to store it in the repository for later use. To enable the user to do this, he or she can send an e-mail to the system, containing the message and any attachments, i.e. the image. This is a way to store content in the repository via the standard e-mail concept.

To trigger the actual checking of the e-mail account and collecting, two approaches shall be used; one user-driven and one via a scheduling mechanism without user interference. Use case 1 and 2 are described in Appendix A.

3.2.3 Use case 3 and 4: Convert RSS feed to AMR and send to e-mail

In a situation where a user wants to listen to the news rather than reading it, a text-to-speech functionality does come in handy. On popular way of reading news on the Internet today is via RSS news feed. In this use case we present a solution for converting a news feed into an audio version using a text-to-speech tool.

In essence, the user wants to convert an RSS news feed to an audio format that can be used on a mobile smart phone, i.e. a Nokia 6630. The audio file shall be sent as an e-mail with the audio file as an attachment. To trigger this event, two options shall be available; one where the user provides the URL to the RSS feed as well as an e-mail account to send it to; and one where the system does this automatically via a scheduler. Use case 3 and 4 are described in Appendix A.

3.2.4 Use case 5: Collect attachment from an email account and send the attachment to a FirstClass conference

FirstClass is often used as an Intranet system on universities around the world. The new versions are full of features and one of them includes sending e-mails to a conference within the system.

In this use case we describe how to collect email messages and attachments from any e-mail account and then send it to a FirstClass conference. This is done by sending an e-mail with the attachment to a predefined e-mail address that is an alias of the actual FirstClass conference.

Please note that this is a scenario where FirstClass handles the user management and CCS acts as a user that sends information to users that have access to the FirstClass conference.

The real use of this particular scenario is when several different schedulers collect content from several different e-mail accounts and then send the collected content to one common delivery channel, in this case a FirstClass conference. Use case 5 is described in Appendix A.
3.2.5 Use case 6: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via e-mail

In this use case we describe how to collect a live feed from a streaming server and send the content to a phone. In our case, the phone can handle 3GPP-files and can receive the file by downloading an e-mail attachment. We also want to store the converted file in the repository for later re-use. This use case enables mobile phones with low bandwidth to experience live content in a re-run manner. Use case 6 is described in Appendix A.

3.2.6 Use case 7: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via the MUSIS news channel

In this use case we describe how to collect a live feed from a streaming server and send the content to a phone. In our case, the phone can handle 3GPP-files and can receive the file by subscribing to specific MUSIS news channel. We also want to store the converted file in the repository for later re-use.

Please note that this is a scenario where the MUSIS system handles the user management and CCS is a content producing middleware that sends information to users that have access to the MUSIS news channel. Use case 7 is described in Appendix A.

3.3 Analysis and design

In this section we will finish this chapter by analyze the use cases and suggest a design based on this analysis and the theory in previous chapters. We start by showing in section 3.3.1 two examples of situations where the original MUSIS approach was not good in the long run. We then elaborate on this in section 3.3.2 and explain our view on the technical approach and implementation that can be used to satisfy the requirements specified in section 3.1.

3.3.1 Design rationale

The following is a list of examples of the MUSIS project when the approach of developing one component that basically contains all functionality was not a good solution;

- During the project new content formats were added. These new formats needed new conversion methods and perhaps new external applications to do these conversions. After a while the MUSIS system became very difficult to upgrade, this due the fact that all functionalities started from the same component, a Java Bean component. It was very hard to locate errors in this component; this because it had became so complex.

- One time one of the external applications on the server crashed and needed to be reinstalled. But some complications also occurred and we had to reinstall the whole server. This task was quite complicated due to the fact that everything used in the MUSIS system was running on this server. So everything had to be reinstalled after the server was reinstalled.

Both these examples show that when a system grows and more and more applications are involved you have to split down functionalities into smaller components that can work independently of each another. This is one argument why the CCS system should have a component-based architecture.

In order to support a high maintainability of our proposed system (see section 3.1.2), we plan to design a system that is flexible enough, so the internal components are easily upgradeable and interchangeable. Thus, we need to use a system design and architecture
model that supports these features. This is another argument why our suggested system should have a component-based architecture.

As mentioned in section 2.2.2 Szyperski defined a software component as follows;

“A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.” (Szyperski 2004)

This means that we have the limitation of not changing the interface between different components, both internal and external. The new system’s main purpose is to handle content and when we analyses the use cases we can divide the system into four different categories of content handling;

- Collect content
- Store content
- Convert content
- Send content

These four categories should be able to work independently of one another but still be coupled as a system. This is described in section 3.2.1 where we define the basic use case model of the CCS system. This is the third, and perhaps the strongest, argument of a component-based architecture.

Metadata is an important part of the CCS system. As seen in the different use cases, the CCS system must be able to store content. As described in section 2.4 content have to be tagged with metadata in order to be easily seek-able and reusable. We also need metadata when we convert content; we need to know the contents format so that we can automatically choose appropriate conversion methods.

We do not have any limit in which types of digital assets the CCS system can store. This means that we do not know how many or which metadata the contents needs to be tagged with. There are also a growing number of standards, and the system should be able to adapt itself to this, for example by allowing new namespaces (see section 2.4.1). The database structure must be implemented with respect to this.

In section 2.5.6 we presented some functionality that was mandatory for the CMS we wanted to use. But due to the fact that none of the CMS we investigated matched all these criteria, we decided to design and implement a new system that will fulfill these criteria. So the new system we want to create needs to have these functionalities mentioned in Table 2.1 - Content Management System Comparison. In the next paragraph we will list these functionalities and how we intend to implement or handle them:

- Content Storing – in a file system on a server. The path of file (content) should be stored as a metadata.
- Metadata management – in a database where different tables can be aggregated so that content and metadata can relate. Metadata also needs to have different namespaces associated with them (see section 2.4.1).
- Plug-in mechanism – components with predefined interfaces that can be added, the system needs to support component based architecture.
- High accessibility to content - a downloading mechanism for remote accessibility, the CCS system should be able to deliver content to different applications/systems running on different platforms.
- Schedule content storing – a scheduler that can perform these tasks via some kind of interface.

- Content conversion – this shall be done by external applications, by doing so we can take the advantage of using already existing and reliable applications.

- Service oriented architecture (SOA) – accessibility with external applications/systems via an application program interface (API). External systems or applications should be able to interact and perform operations on the CCS system via the Internet.

Figure 3-2, describes the generic architecture of the proposed system. The figure is based on the requirements and criteria that have emerged during the design rationale in this section.
As seen in Figure 3-2, the system is based on several different input and output mechanisms. Therefore, the system should be scaleable (Jansen et al., 2005). By this, we mean that the system should consist of modular, reusable and easily expandable components to be able to deal with new types of content. This includes the whole system, i.e. the collecting, storing, converting and the sending mechanisms.

Figure 3-2 consists of four different domains, the Internet (no. 1 in figure above), the CCS system (no. 2), Mobile devices (no. 3) and External user management system (no. 4).

- The Internet (no. 1) is where the CCS system collects its content. This content can be of different types such as text, audio, images and videos. These types can
be in different formats such as RSS (text), Mp3 (audio), jpeg (image) and Mpeg (video).

- The CCS system (no. 2) consists of various components, both components that are developed especially for this system and components that already exist. The components can be categorized into four major domains, Collect, Store, Convert and Send:
  - Collect – have the responsibilities of collecting content from the Internet
  - Store – shall have the capabilities of storing both content and metadata that are related to this content.
  - Convert – this domain takes care of the actual converting of content. It must be upgradeable i.e. a mechanism for adding new conversion methods must developed.
  - Send – this is the output channel in the CCS system. The output channels should be in different formats or methods and the system also should have the capability to add new output methods, Examples of output channels are; RSS feeds, SMS messages, email and podcasting.

- Mobile devices (no. 3), these are the main receivers of the converted contents.

- External content management systems (no. 4) have the responsibilities of handling the user management. By this we mean that the CCS system does not contain any sophisticated mechanism for user management so you need to use existing external ones.

Please note that the CCS system also have components for Management. This is how you interact with the CCS system and this shall be done both manual and automatic.

This analysis leads us forward to the technical approach that will be described in the next section.

### 3.3.2 Technical approach

In this section we present our approach regarding the software solutions and how existing software components are used or connected to the system.

Based on the discussions in the previous section 3.3.1 we have come to the conclusion that we need an architecture that is component based and platform independent. After looking into the different component systems that are listed in section 2.2.3 we decided that we shall use Web services to achieve these goals. Web services are platform independent and the other listed are not, except for CORBA and EJB’s. However, CORBA and EJB may be still necessary to implement sophisticated back-end services:

> [...] Web Services come into play when these islands must be connected to full-blown networked systems. [...] (Stal 2002)

CORBA is also known as being complex to implement when comparing to Web services and Enterprise JavaBeans (EJB). EJB with RMI are language dependent, but Web services can be implemented in any programming languages.

By choosing Web services we can also take advantage of having a Service Oriented Architecture (SOA), see section 2.3.1 for more information about SOA.

Another conclusion in section 3.3.1 was that the CCS system needs to have accessibility with external applications/systems by using a service-oriented architecture (SOA). By using Web services we open up the CCS system for external
applications/systems to interact with the functionalities that the CCS system provides. This can not be realised using only Enterprise JavaBeans, for example.

In section 3.2.2 we have the non-functional requirements portability and reliability, so we need a programming language that supports these requirements. This section also points out that the language we are going to use need to support a component-based architecture. And the language should also support Web services as we have decided to use this in the CCS system.

Java is a platform independent programming language that supports Web services. One of Java’s techniques is JavaBeans and this technique supports component-based architecture, below is Sun’s definition of JavaBeans;

“JavaBeans technology is the component architecture for the Java 2 Platform, Standard Edition (J2SE). Components (JavaBeans) are reusable software programs that you can develop and assemble easily to create sophisticated applications” (SunMicrosystems 2006b)

One of Java’s editions is described in the citation below;

“Java Platform, Enterprise Edition (Java EE) is the industry standard for developing portable, robust, scalable and secure server-side Java applications. Building on the solid foundation of Java SE, Java EE provides web services, component model, management, and communications APIs that make it the industry standard for implementing enterprise class service-oriented architecture (SOA) and Web 2.0 applications.” (SunMicrosystems 2006a)

When using Java EE as the programming language we will have a tool that support the demands that came up in the analysis (see section 3.3.1). As we have already decided we shall not use EJB’s, which is an extension of JavaBeans, as an interface for communication with external components. However, we could use JavaBeans within the components and use Web services as the interface for external communication. The basic functionality in JavaBeans is enough for the implementation of the CCS internal component system. The main reason for not using EJB’s as an external component interaction method is the limitation to communicate with other EJB’s.

We have chosen MySQL²⁹ to act as the database for the CCS system; MySQL is a relational database that has enough capacity for the prototype of the CCS system. MySQL can also be installed on a lot of different platforms.

To be able to transfer files between different computers within CCS, we want to use a technique that is common and that can be implemented on a large variety of servers and architectures. There should also be API’s for the technique in the most common programming languages. Thus, our choice fell on the File Transfer Protocol, FTP.

The Web services and the web applications developed in this thesis need to be run on an application server. Since the Sun Java System Application Server Platform Edition 8.2³⁰ is a common application that can be installed in common architectures, and also free of charge to use, this was a natural choice.

We will use IDE’s that support the proposed techniques and language. Our choice is the NetBeans IDE³¹. The most important feature in the scope of this thesis the support for Web services, which NetBeans handle with great confidence. IT also supports deployment to common application server, including Sun Java System Application Server Platform Edition 8.2, which we use on one of the servers. We also use the application server used in the Glassfish project³², which is built upon Sun Java System Application Server Platform Edition 9.0, which is Java EE 5 compatible.
To be able to make a demonstration web site that uses the Web services we develop, we use Sun Java Studio Creator 2\textsuperscript{33}. In Studio Creator 2 you have built in web service support and a component based way of working with the code. Please note that the use of the Java programming language or a component based way of using the web services is not necessary in order to use the CCS system.

Section 3.4.1 and section 3.4.2 gives us a design and a technical approach on how to implement the CCS system, this implementation is described in chapter 4.
4 Technical Implementation, Scenarios and Assessment

In this chapter we will describe how we have implemented the proposed suggestion made in the previous chapters. We will start in section 4.1 by describing the different components that are involved in the CCS system. In section 4.2 we talk about the software components that we have implemented in the CCS system.

By using sequence diagrams we try to in more detail explain how the different components works together this is done in section 4.3. We finish this chapter by describing how to add a new conversion method into the CCS system; this is done in section 4.4.

4.1 Software and servers

In this section, we will illustrate where the components are deployed and which servers that are used in the CCS system and Figure 4-1 illustrates this. Please note that this is the technical perspective on Figure 3-2. We use the four identified domains (Collect, Store, Convert and Send) and mark these using a bold italic style. For a complete list of software and servers used in CCS, see Appendix B.

As shown in Figure 4-1 below, we have implemented the CCS system on two different servers; Dellserv a Linux server (no. 1) and Zebra a Mac OS X server (no. 2). The user is interacting with the system via a Web-interface, and in the figure above it runs in a browser on a PC with Windows XP as operating system (no. 3). The Web services can be initialized using the automatic scheduler, which is a separate Java Application that can run on any Java enabled client (no. 4).

On Dellserv a Sun Java System Application Server is running (no. 5) some of the web services are running on this application server. The GUI which is implemented as a web interface with JSF technology is also running on this application server on Dellserv. The CCS repository database and a FTP-server are also situated on this server. The FTP- server is used for file transaction between the different parts of the system.

Zebra (no. 2) has a Sun Java System Application Server (no. 6), this application server hosts the rest of the web services. Zebra also has a FTP-server for file transaction and on Zebra there are a number of applications that are used for different conversions and/or manipulations of contents.

The user connects to the system via a web interface that is running on Dellserv as a JSF application. On Dellserv, the Cocoon server (no. 7) for converting text is also running.
In the next chapter we will describe the different web services that are running in the CCS system.

4.2 Software components
To be able to fulfill the demands we discovered in our design phase the system needs to split up into different components. We will use web services and Java Beans technologies to achieve this.

In Figure 3-2 and Figure 4-1, we identified and described four different domains that we need in the CCS system:

- Collect
- Store
- Convert
- Send
The Java Beans that the web services are using are described in Appendix C. The GUI and Scheduler are described in Appendix D. The MySQL database (no. 8 in Figure 4-1) and the tables that handle the metadata relations are described in Appendix E.

### 4.2.1 Web services used in the CCS system

In table 4-1, we have listed all the Web services we have implemented and the methods they provide. You can also see to which part of the system the different methods belong: collect, store, convert or send. The name in parenthesis is the name of the server where the Web services are deployed (zebra or dellserv).

<table>
<thead>
<tr>
<th>Web services</th>
<th>Collect</th>
<th>Store</th>
<th>Convert</th>
<th>Send</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSWebServices (zebra)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>collectContent()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checkConvertFormats()</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>convertContent()</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CCSMailWebService (dellserv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uploadFromEmail()</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>sendEmailWithAttachment()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCSRepositoryWebService (dellserv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uploadToRepositoryFromFTP()</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>addObjectToRepository()</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CCSSendRSSWebService (dellserv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>repositoryIDToRSS()</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CCSVASPWebService (dellserv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sendContentToMUSIS()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1 - The Web services methods

The different Web services we have developed and a short description of the different methods they will provide are:

- CCSWebServices
  - collectContent()
    - Downloads content in different ways.
  - checkConvertFormats()
    - Checks a database table to see which suitable formats the CCS-system can convert a given content.
  - convertContent()
    - Converts content to selected formats

- CCSMailWebService
  - uploadFromEmail()
    - Uploads content that are attached to an email.
  - sendEmailWithAttachment()
    - Sends content as an attachment via an email.

- CCSRepositoryWebService
  - uploadToRepositoryFromFTP()
- Uploads content to the CCS repository from a given FTP
  - `addObjectToRepository()`
    - Adds an object to the repository, no file is uploaded.
- CCSSendRSSWebService
  - `repositoryIDToRSS()`
    - Creates an RSS feed with enclosure.
- CCSVASPWebService
  - `sendContentToMUSIS()`
    - Sends content via a MUSIS channel.

In the next chapter we will describe in more detail how the different components interact this will be done by using sequence diagrams.

### 4.3 Use case analysis (UC4) using UML sequence diagrams

To make this thesis more readable we have decided not to explain in more detail all of the components we have implemented. Instead we will describe one of the use cases in more detail. We have chosen the use case with the id UC 4 (see section 3.2.3), this use case interacts with almost every functionality in our system. This use case is about how the scheduler automatic convert a RSS feed to an AMR file and sends this file as an attachment in an e-mail.

We will use sequence diagrams to show how the different components interact with each another. In section 4.4 we will describe how you can manually convert content and how the CCS system can be expanded with new conversion methods.

![Figure 4-2 Component interaction](image-url)
This first diagram, Figure 4-2, shows how all of the Web services that are used in this use case interact with each another.

The CCSScheduler makes a call to the CCSWebService (CCSWS) and invokes the method collectContent. This method takes the RSS feed that are send in as an argument to the method and downloads it and stores it as a file. This file is then uploaded to the CCS repository (CCSRepositoryDB) with help of the CCSRepositoryWebService. After this procedure the CCSScheduler calls another method in the CSWS, the convertContent method. By sending in right conversion method as an argument the content is converted. Then the converted content is uploaded to the CCS repository and sent as an attachment to an email to the appropriate user. The last step is done by using the CCSMailWebService. In the next sequence diagrams we will describe each of these steps in more detail and which beans are used by the different Web services.

We have three Web Services that are involved in this use case; CCSWebServices, CCSRepositoryWebService and CCSMailWebService. Both CCSWebServices and CCSRepositoryWebService are called twice but CCSRepositoryWebService does the same thing twice so we just show you this once.

In figure 4-3, the CCSScheduler calls the method collectContent in the web service CCSWS with an argument. This argument is the URL to the requested RSS-feed. CCSWS uses the Java Bean CCSFileBean and the method downloadContentURL in this bean to perform this task. The downloaded content is stored as a file on a FTP-server. We choose to use FTP for file transactions, this because FTP is widely used for file transportation on the web so its easy to expand the CCS system or get files from other systems which can provide a FTP server.
Figure 4-4 Repository storing with automatic metadata generation.

After the RSS feed has been downloaded we need to store this file in the repository with appropriate metadata. This is done with the CCSRepositoryWS and its method uploadToRepositoryFromFTP. This method has a lot of parameters; see the interface for this method:

```java
public int uploadToRepositoryFromFTP(String fileName, String[] dcUserMD, String ftpHost, int ftpPort, String ftpUsername, String ftpPassword, String ftpMode, String ftpServername) throws java.rmi.RemoteException;
```

The first parameter is the name of the file that you want to upload, the string array dcUserMD is an array of all metadata that is manually set by the user or automatically by the user interface. The array is built up as follows the first position in the array is the name of the metadata ex. file format and the second position is the value of the metadata ex jpeg. The array continues like this, first name then value of metadata. By doing so we do not need to know how many metadata tags this content will have in advance and we do not need to know or specify which metadata the content needs to have. By using this approach the CCS system is very open in that sense that you can send in any and how many metadata tags with value as you which.

All the other parameters are data about the FTP server which holds the content you are about to upload to the CCS repository. You can see that they all have ftp as a prefix. So you are not stuck with one FTP server, the data is sent in when you are uploading. It’s not in any configuration file nor hard coded in the web service. We have tried to have this approach during the whole implementation phase, to make the CCS system as open as possible is important if you want the system to be easy or even possible to expand and upgrade.

As you can see in the diagram Figure 4-4, the method uploadToRepositoryFromFTP calls a method in the CCSAddToRepositoryBean, downloadFileFromFTP. This method takes all the data about the FTP server such as address, username, port number etc as parameters and downloads the content to the repository.

The rest of the beans and their methods in this diagram (Figure 4-4) handle with metadata extraction. Both metadata that are sent in via the array from the user and
metadata that are automatically extracted from the content are stored in the database. The CCS system can extract metadata that are specific for these file types; images, video and audio files. It also extracts some basic metadata from any content such as file size, date created etc. If the collected file is an XML file, the root element will often be of value for later conversions and as a guide to what content that can appear in the file. For example, an RSS news feed can be recognized in this way. This can be helpful since RSS feeds often use the file extension “.xml” and not “.rss”. This is true for many xml file types. The automatic metadata extraction process also incorporates Mplayer and its video and audio functions. For example, Mplayer can extract the frame rate, codec and the frame size of a video clip. This is essential metadata when it comes to converting a video file to another format, e.g. to a format that suits a mobile phone or an iPod.

When all the metadata is extracted the system stores this data in a database. The metadata is also bound to the content; this is described in Appendix E.

Now we have uploaded the RSS feed as a file to the repository and stored metadata about the content in the database. The next step is to convert the RSS feed into an audio-file. We need to do this in three different steps, first we convert the RSS feed into a text file, then we convert the text to audio and last we convert the audio file once more to another audio format. We need to do this last step because the application we use to convert text to audio converts to another format than we want.

For all these conversions we use external applications that both are running on the server and applications that are running on remote servers. We can use any application on the server that is command-line controlled.
In the first step we use a Cocoon server to do the conversion from RSS feed, the Cocoon server is running on another server. If you are interested in how a Cocoon server works please look at the official Apache Cocoon website.

For the second step, text to audio conversion, we use a built in application on the Mac OS X server. The name of this application is say, and you just send in the path of the text-file and the path of the audio-file to this application to make this conversion.

The third and last step, the conversion from the aiff-format into the amr-format, is using the application ffmpegX. You can also see the source code in Appendix F, for how we used ffmpegX in this conversion.

After the content is converted we need to upload the converted content to the repository and store appropriate metadata about this content into the database. This is done in the same way as we described in Figure 4-4. So we refer to this diagram instead of writing the same thing twice.

In Figure 4-6, we will show how the CCS system sends the converted content as an attachment in an email to appropriate users.

The process shown in Figure 4-6 starts with that CCSScheduler calls the method sendEmailWithAttachment in CCSMailWS. This method takes two parameters, the first is a string-array which will hold all the email account data and the second is the path to the content you want to send. CCSMailWS calls the method downloadFromRep in CCSFileBean to be able to temporarily store the content that are about to be sent.

When the content has been downloaded CCSMailWS then calls the method sendEmailAttachFile in CCSEmailBean and passes on all the parameters. This method takes care of sending the email and attaches the content to the email. This ends this procedure, the scheduler have downloaded a RSS feed and converted it into an audio file. This converted file was then sent as an attachment via an email to appropriate users.
In Figure 4-7, we will illustrate how the different components of the CCS system are interacting and where they are situated.

This Figure 4-7 does not differentiate so much from Figure 4-1, we have more specific named the different Web services that are in use in this user case and named the different applications that are used for the conversions that take place. In the next chapter we will explain how the CCS system deals with the issue of adding a new conversion method.

4.4 How to add a new conversion method without changing anything in the client side

In section 4.3 we described most of the functionalities in the CCS system but not all were included. In this section we will show how we have solved one of the problems with having a component-based architecture; the problem of keeping the components interface unchanged when you need to expand the system. As described in section 2.2.2 a component is according to Szyperski a unit that have a “contractually specified interfaces” (Szyperski 2004)

We can not change the interface, it needs to be the same, and otherwise the user of the component also needs to change their applications/systems when something is added to the system. We have chosen Web services as the interface between the different components in the CCS system (see section 3.3.1).

So how shall we do when we want to add more conversion methods to the system, we can not just add some more methods in the web service CCSWebServices (see section 4.2.1). If we do so we change the interface for this method. Instead we send the
name of the conversion as an argument to one method, the method convertContent. By doing so we can add how many new conversions we want without changing the interface for this web service. So now a user just needs to know which conversion methods the CCS system can provide for a specific content. We solved this by first the user calls the method checkConvertFormats and then convert the content with method convertContent. Both these methods belong to the web service CCSWebServices. The sequence diagram shown in figure 4-8 describes how the method checkConvertFormats interact with other components.

![Sequence Diagram](image)

The method checkConvertFormats first checks the format of the content that the user wants to convert. Any content that needs to be converted has to be uploaded first to the CCS repository; this is done so that the CCS system can keep a history record of all the converted content and trace the original content with all its metadata. For example the original content URL can be of importance. One of the metadata that is automatically produced when you upload content to the CCS system is the contents format.

When the format is established the method checks against the database if the CCS system provides with any conversions methods for this format. If there are any the user gets all the conversions suggestions back. The method checkConvertFormats returns this result as a string array. The convert process is described in detail in Figure 4-5.

The user then has to choose which conversion method to use and then send the name of this method as an argument to the method convertContent. If the CCS system does not provides with a suitable conversion method the user gets back a notification about this. The user then has to inform the administrator of the CCS system that this conversion method is missing. The administrator then can upgrade the CCS system with this conversion method. This is done by adding a new method in the Java Bean CCSConvertBean that performs this conversion. The database also needs to be updated with the name of this new conversion alternative.

The user then just needs to perform the same procedure as before and then get the new conversion for the content without having to change anything in his application/system.
5 Discussion
This section concludes the thesis by relating to the thesis goals and by discussing our contribution to the field of research. We also elaborate on future developments regarding the CCS system. Finally, we present our conclusions.

5.1 Did we reach our goal?
The thesis goal is as follows (see section 1.2):

*How can content already available in the Internet be re-used and how can we design a system that supports data portability and media migration?*

We also limited this broad question into the MUSIS context, with the consequence of narrowing it down to a specific number of requirements that can be fulfilled within the scope of this thesis.

In our opinion, we have reached our goal mentioned above. We did this by designing a system that can handle the requirements found in section 3.1. The system is designed in a flexible manner with a strong emphasis on services and an architecture that supports a component based system. We solved this by using web services and a programming language that supports a component based architecture.

Four different domains were identified and implemented within the CCS system, showing that the system can be separated but still seen as one.

In section 2.5 we identified some important functionality that the CCS system needs to have in order to reach the thesis goal. We can look at that table once more and add CCS as a new column:

<table>
<thead>
<tr>
<th></th>
<th>Apache Lenya</th>
<th>PostNuke</th>
<th>MMBase</th>
<th>FirstClass</th>
<th>CCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Storing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Metadata management</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Automatic metadata generation</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Plug-in mechanism</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>High accessibility to content</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Schedule content storing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Content conversion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Service oriented architecture (SOA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5-1 Content management system comparison - with CCS*

As seen in table 5-1, we believe that the CCS system is able to fulfil the criteria’s mentioned. Please note that this is within the MUSIS context and that the CCS system should not be seen as a replacement for any of the above mentioned content management systems.
5.2 What is our contribution to the field?
We have accomplished to make a system that has some innovative functionality compared to the above mentioned systems. We would like to highlight three different areas:

- **Content conversion via a Web services interface**
  - The CCS system is able to convert content in a flexible way via a Web service interface. With this approach, any system with Web services capabilities and have a need for converting content can use the CCS system to achieve this. The conversion process can therefore be scheduled and done automatically or manually via a graphical user interface.

- **Flexible plug-in mechanism by using a component-based architecture**
  - Within the CCS system, we use a component-based architecture that allows us to add new functionality by only changing the affected component and not the whole system. For example:
    - Add a collect mechanism via FTP
    - Convert DVD movie to iPod video
    - Send RSS news as SMS

- **Easily interact with external applications or systems**
  - Collecting content with the CCS system can be done by:
    - manually uploading the content from the user via a GUI
    - downloading content from a streaming server
    - collect content via HTTP
  - All conversion processes are made by external applications or systems, for example:
    - Cocoon, RSS feed to pdf/rtf/html/txt
    - Mplayer, mencoder and ffmpegX, RTSP to 3GP
    - ImageMagick, scale JPEG images
    - Mac OS X Speech Synthesize module, Say, text to aiff audio format

- **All process in the system could be done manually or automatically**
  - This is done by enabling several Web service interfaces to the system, allowing other applications or system to interact with the CCS system. This could be done automatically via a scheduler, or manually via a GUI.

5.3 What should we have done differently?
We could have put more effort and time in the database design and the distributed computing architecture. The system is allowing a flexible way of inserting the metadata into the database, but at the cost of complicated and time-consuming searches. The server park could be larger, allowing us to dedicate servers to different components in the system.
5.4 Future development
So far, we have been able to use the CCS system in one project, the AMULETS project. The project can be described using the following citation:

“The paper describe our current efforts related to the design and technical aspects of implementing a particular ubiquitous learning environment with school children to integrate systems supporting alternative ways of interaction to mediate social aspects of learning, knowledge construction, educational gaming and reflection.” (Kurti et al. 2006)

The AMULETS project was satisfied with the CCS system and will use it in upcoming activities during the fall 2006. The project will especially focus on how the consumer can contribute with content that can be re-used in the same event, allowing the user to be a prosumer. The CCS system will be the mediator of the content that is collected, converted and sent, all within the same activity or even a single event. One area of development in this particular project is SMS messages as a sending mechanism.

The CCS system is also planned to be used in the sequel of MUSIS, the MUSIS 2 project. The project will continue to explore mobile services and how content can be experienced to small devices, both as a consumer and a producer. The MUSIS 2 project is currently designing the different scenarios and what parts of the CCS system that is going to be used.

As seen here, the CCS system will be used in different projects in a near future, allowing more time and effort to be put into the development and evaluation of the system. One future focus area will be on automatic content classification using ontologies. Another part of the system that will be extended is the sending mechanism, for example via SMS or MMS.

5.5 Conclusions
If we consider Figure 2-1 in section 2.1 again, we can see the three areas that have been the theoretical foundation of the CCS system. We have built the system using software components and metadata that interoperability can be relied upon:

- Software components, the whole CCS system is built using a software component architecture. In the design chapter we found that maintainability using internal components that are easily upgradeable and interchangeable is important factors for achieving our goals.
- Metadata is extracted and generated when collecting content. The metadata is then used in the system to support content selection, conversion and sending.
- Interoperability between different content formats is supported via the conversion process. Interoperability with internal or external systems is supported via Web services. This allows other systems to use functionalities within the CCS system in a standardized way.

We have shown in this thesis how we have designed and implemented a system with new innovative features regarding data portability and media migration. Although it is design for a specific context, the functionalities could easily be extended to other domains. This is due to the service oriented architecture for the external communication and interaction. It is also thanks to the component-based architecture in the system.

We believe that there is a need for the CCS system within the mobile market. As market grows rapidly, the issues related to data portability and media migration will also grow.
6 References


Balzer, Y. 2004, *Improve your SOA project plans*, IBM, viewed 2006-08-28


Cox, B. 1992, 'What if there is a Silver Bullet and the competition gets it first?' *Journal of Object-oriented Programming*.


FirstClass 2006, *About FirstClass*, Open Text Corporation, viewed 2006-09-01

<http://dx.doi.org/10.1109/2.410146>.


Fukumoto, T. 2000, 'The Criteria and Evaluation of Metadata/Keywords in Image Retrieval',


<http://dx.doi.org/10.1109/MPRV.2004.1321024>.


<http://dx.doi.org/10.1002/0471703567>.


7 List of tables

TABLE 2-1 - CONTENT MANAGEMENT SYSTEM COMPARISON ................................................................. 21
TABLE 4-1 - THE WEB SERVICES METHODS ...................................................................................... 35
8 List of figures

FIGURE 2-1 THE THREE PART INTERPLAY ................................................................. 10
FIGURE 2-2 CONVERSION OF CONTENT .............................................................. 11
FIGURE 3-1 BASIC USE CASE MODEL ................................................................. 24
FIGURE 3-2 GENERIC ILLUSTRATION OF CCS .................................................. 29
FIGURE 4-1 THE COMPONENTS IN THE CCS SYSTEM ........................................ 34
FIGURE 4-2 COMPONENT INTERACTION ............................................................. 36
FIGURE 4-3 COLLECTING AN RSS-FEED ............................................................ 37
FIGURE 4-4 REPOSITORY STORING WITH AUTOMATIC METADATA GENERATION 38
FIGURE 4-5 CONTENT CONVERSION ................................................................. 39
FIGURE 4-6 SENDING CONTENT VIA E-MAIL .................................................... 40
FIGURE 4-7 COMPONENT INTERACTION IN USE CASE 4 .................................. 41
FIGURE 4-8 DYNAMIC CONVERSION METHOD CHECKING ................................ 42
FIGURE 5-1 CONTENT MANAGEMENT SYSTEM COMPARISON - WITH CCS .......... 43
9 List of footnotes

4. http://www.w3.org/TR/xml/, viewed 2006-10-05
5. http://www.w3.org/TR/soap/, viewed 2006-10-05
7. http://www.w3.org/TR/wsd1, viewed 2006-10-05
24. MP3 – MPEG Layer 3, Audio codec defined by Motion Pictures Experts Group
25. AMR – The audio codec used in the 3GPP format
26. RM – RealMedia format
27. Real Time Streaming Protocol - Protocols for streaming servers
28. 3rd Generation Partnership Project – Project focusing on mobile media related technologies
32. https://glassfish.dev.java.net/, viewed 2006-10-05
34. http://cocoon.apache.org/, viewed 2006-10-05
### Appendix A – Use cases

In section 3.1, we saw that the requirements of the system demand a complex system with many components and mechanisms. In this Appendix, we try to illustrate the design of the system by showing use case models that are described in a more narrative form in section 3.2.2 to 3.2.6.

**Use case: Collect content from e-mail with predefined email account**

**ID:** UC1

**Actors:**
User

**Preconditions:**

**Flow of events:**
1. The user enters the web page of collect content by email
2. The user selects a predefined email account
3. The user clicks the Collect now-button
4. The system activates the email-collector component
5. If there are any new emails with content to collect
   5.1. The system extracts the content and uploads it to the repository
   5.2. The user is notified
6. If there are not any new emails with content to collect
   6.1. The user is notified

**Post conditions:**
1. Any new email messages and attachments have been uploaded to the repository

---

**Use case: Scheduled collect content from e-mail with predefined email account**

**ID:** UC2

**Actors:**
Scheduler
Time

**Preconditions:**
1. A scheduled event that triggers this action exists
2. The email account to check is specified

**Flow of events:**
1. Time activates Scheduler
2. The Scheduler activates the system email-collector component with email account to check
3. If there are any new emails with content to collect
   3.1. The system extracts the content and uploads it to the repository
4. If there are not any new emails with content to collect
   4.1. The use case is terminated

**Post conditions:**
1. Any new email messages and attachments have been uploaded to the repository
### Use case: Convert RSS feed to AMR and send to e-mail with manual settings

<table>
<thead>
<tr>
<th>ID: UC3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors:</strong></td>
</tr>
<tr>
<td>User</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>1. The user knows the URL to the RSS feed, the format that he or she want to convert it to and the email address to send it to</td>
</tr>
<tr>
<td><strong>Flow of events:</strong></td>
</tr>
<tr>
<td>1. The user enters the web page of collect and convert content</td>
</tr>
<tr>
<td>2. The user enters the URL to the RSS feed</td>
</tr>
<tr>
<td>3. The user clicks the Collect and Store-button</td>
</tr>
<tr>
<td>4. The system collects the content and stores it in the repository</td>
</tr>
<tr>
<td>5. If the collecting and storing is not successful</td>
</tr>
<tr>
<td>5.1. The user is notified</td>
</tr>
<tr>
<td>5.2. The use case is terminated</td>
</tr>
<tr>
<td>6. If the collecting and storing is successful</td>
</tr>
<tr>
<td>6.1. The user clicks the Convert-button</td>
</tr>
<tr>
<td>6.2. The system decides on which conversion options to present</td>
</tr>
<tr>
<td>6.3. The user chooses the RSS-to-AMR-option</td>
</tr>
<tr>
<td>6.4. The user clicks the Do conversion-button</td>
</tr>
<tr>
<td>6.5. The system activates the conversion component</td>
</tr>
<tr>
<td>6.6. If the conversion is not successful</td>
</tr>
<tr>
<td>6.6.1. The user is notified</td>
</tr>
<tr>
<td>6.6.2. The use case is terminated</td>
</tr>
<tr>
<td>6.7. If the conversion is successful</td>
</tr>
<tr>
<td>6.7.1. The system stores the converted AMR-file in the repository</td>
</tr>
<tr>
<td>6.7.2. The user clicks the Send converted content-button</td>
</tr>
<tr>
<td>6.7.3. The user enters the e-mail address to send the AMR-file to</td>
</tr>
<tr>
<td>6.7.4. The user clicks the Send email-button</td>
</tr>
<tr>
<td>6.7.5. If the sending of the e-mail is successful</td>
</tr>
<tr>
<td>6.7.5.1. The user is notified</td>
</tr>
<tr>
<td>6.7.5.2. The use case is terminated</td>
</tr>
<tr>
<td>6.7.6. If the sending of the e-mail is not successful</td>
</tr>
<tr>
<td>6.7.6.1. The user is notified</td>
</tr>
<tr>
<td>6.7.6.2. The use case is terminated</td>
</tr>
<tr>
<td><strong>Post conditions:</strong></td>
</tr>
<tr>
<td>1. The content is collected and converted and stored in the repository, converted to an AMR file which is sent to an email address</td>
</tr>
</tbody>
</table>
**Use case: Convert RSS feed to AMR and send to e-mail with predefined settings**

**ID:** UC4

**Actors:**
Scheduler  
Time

**Preconditions:**
1. A scheduled event that triggers this action exists  
2. The URL and what conversion to make is specified as well as the email address to send the converted file to

**Flow of events:**
1. The scheduler activates the collector component with the URL  
2. The system stores the collected content in the repository  
3. The scheduler activates the conversion component with the appropriate content and conversion type  
4. If the conversion is successful  
   4.1. The system stores the converted content in the repository  
   4.2. The system sends the converted content to the specified email address  
   4.3. If the sending is successful  
      4.3.1. The use case is terminated  
   4.4. If the sending is not successful  
      4.4.1. The use case is terminated  
5. If the conversion is not successful  
   5.1. The use case is terminated

**Post conditions:**
1. The content is collected and converted and stored in the repository, converted to an AMR file which is sent to an email address
<table>
<thead>
<tr>
<th>Use case: Collect attachment from an email account and send the attachment to a FirstClass conference with predefined settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID:</strong> UC5</td>
</tr>
<tr>
<td><strong>Actors:</strong> Scheduler, Time</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>1. A scheduled event that triggers this action exists</td>
</tr>
<tr>
<td>2. Use case UC2 is successfully terminated</td>
</tr>
<tr>
<td>3. The email account to send the content to is specified</td>
</tr>
<tr>
<td><strong>Flow of events:</strong></td>
</tr>
<tr>
<td>1. The system sends the collected content to the specified email address</td>
</tr>
<tr>
<td>2. If the sending is successful</td>
</tr>
<tr>
<td>2.1. The use case is terminated</td>
</tr>
<tr>
<td>3. If the sending is not successful</td>
</tr>
<tr>
<td>3.1. The use case is terminated</td>
</tr>
<tr>
<td><strong>Post conditions:</strong></td>
</tr>
<tr>
<td>1. The attachment has been sent via an e-mail to the FirstClass Conference</td>
</tr>
</tbody>
</table>
## Use case: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via e-mail with manual settings

### ID: UC6

### Actors:
User

### Preconditions:
1. The user knows the URL to the RTSP feed, the format that he or she want to convert it to and the email address to send it to

### Flow of events:
1. The user enters the web page of collect and convert content
2. The user enters the URL to the RTSP feed
3. The user clicks the Collect and Store-button
4. The system collects the content and stores it in the RealMedia format
5. If the collecting and storing is not successful
   5.1. The user is notified
   5.2. The use case is terminated
6. If the collecting and storing is successful
   6.1. The user clicks the Convert-button
   6.2. The system decides on which conversion options to present
   6.3. The user chooses the RealMedia-to-3GPP-option
   6.4. The user clicks the Do conversion-button
   6.5. The system activates the conversion component
   6.6. If the conversion is not successful
      6.6.1. The user is notified
      6.6.2. The use case is terminated
   6.7. If the conversion is successful
      6.7.1. The system stores the converted 3GPP-file in the repository
      6.7.2. The user clicks the Send converted content-button
      6.7.3. The user enters the e-mail address to send the 3GPP-file to
      6.7.4. The user clicks the Send email-button
      6.7.5. If the sending of the e-mail is not successful
         6.7.5.1. The user is notified
         6.7.5.2. The use case is terminated
      6.7.6. If the sending of the e-mail is successful
         6.7.6.1. The user is notified
         6.7.6.2. The use case is terminated

### Post conditions:
1. The content is collected and stored in the repository, converted to an RGPP file which is stored in the repository and then sent by e-mail
<table>
<thead>
<tr>
<th>Use case: Collect a streaming RTSP feed and send it to the phone as a 3GPP-file via the MUSIS news channel using predefined settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: UC7</td>
</tr>
<tr>
<td><strong>Actors:</strong></td>
</tr>
<tr>
<td>Scheduler</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>1. A scheduled event that triggers this action exists</td>
</tr>
<tr>
<td>2. The URL to the RTSP feed and what conversion to make is specified as well as the MUSIS channel name to send the converted file to</td>
</tr>
<tr>
<td><strong>Flow of events:</strong></td>
</tr>
<tr>
<td>1. The scheduler activates the collector component with the URL</td>
</tr>
<tr>
<td>2. The system stores the collected content in the repository</td>
</tr>
<tr>
<td>3. The scheduler activates the conversion component with the appropriate content and conversion type</td>
</tr>
<tr>
<td>4. If the conversion is successful</td>
</tr>
<tr>
<td>4.1. The system stores the converted content in the repository</td>
</tr>
<tr>
<td>4.2. The system sends the converted content to the specified MUSIS channel</td>
</tr>
<tr>
<td>4.3. If the sending is successful</td>
</tr>
<tr>
<td>4.3.1. The use case is terminated</td>
</tr>
<tr>
<td>4.4. If the sending is not successful</td>
</tr>
<tr>
<td>4.4.1. The use case is terminated</td>
</tr>
<tr>
<td>5. If the conversion is not successful</td>
</tr>
<tr>
<td>5.1. The use case is terminated</td>
</tr>
<tr>
<td><strong>Post conditions:</strong></td>
</tr>
<tr>
<td>1. The content is collected and stored in the repository, converted to a 3GPP file which is stored in the repository and then sent to a MUSIS channel</td>
</tr>
</tbody>
</table>
Appendix B – Software and servers in the CCS system

The software we used when we implemented the system:

- Sun NetBeans IDE 5.0\(^1\)
  - Developing the Web services in Java
  - Developing java application

- Sun Java Studio Creator 2.0\(^2\)
  - Developing the GUI with JSF-pages (Java Server Faces\(^3\))

The servers and what is running on them:

- Dellserv (see no. 1 in Figure 4-1) – a Linux server running Kubuntu\(^4\)
  - Sun Java System Application Server\(^5\)
    - CCS Web services
    - CCS GUI – JSF pages
    - MySQL\(^6\) database
  - File system
    - CCS Repository
  - Apache Cocoon server\(^7\) – text conversion

- Zebra (see no. 2 in Figure 4-1) – a Mac OS X server\(^8\)
  - Sun Java System Application Server\(^5\)
    - CCS Web services
  - Applications
    - ffmpeg\(^9\) – converting video
    - Mplayer OS X\(^10\) – converting video and audio
    - MEncoder\(^11\) – converting video
    - Mac OS X Speech Synthesis manager (Say)\(^12\) - text to speech
    - ImageMagick\(^13\) – converting images

---

\(^1\) http://www.netbeans.org/, viewed 2006-10-05
\(^2\) http://developers.sun.com/prodtech/javatools/jscreator/index.jsp, viewed 2006-10-05
\(^3\) http://java.sun.com/javaee/javaserverfaces/, viewed 2006-10-05
\(^4\) http://www.kubuntu.org/, viewed 2006-10-05
\(^5\) http://www.sun.com/software/products/appsrvr/index.xml, viewed 2006-10-05
\(^6\) http://www.mysql.com, viewed 2006-10-05
\(^7\) http://cocoon.apache.org/, viewed 2006-10-05
\(^8\) http://www.apple.com/server/macosx/, viewed 2006-10-05
\(^9\) http://homepage.mac.com/major4/, viewed 2006-10-05
\(^10\) http://mplayerosx.sourceforge.net/, viewed 2006-10-05
\(^11\) http://www.mplayerhq.hu/, viewed 2006-10-05
\(^12\) http://www.hmug.org/man/1/say.php, viewed 2006-10-05
\(^13\) http://www.imagemagick.com/, viewed 2006-10-05
Appendix C – JavaBeans used in the CCS system

All the Web services are built up from reusable components, we have used both Java Beans that we have implemented and applications/services that we have not. This is one of the advantages when you are using software component architecture. You can use and reuse components that you have not implemented yourself.

In the next section we will list all the Java Beans we have implemented and shortly what their responsibilities are.

They all belong to a java package called se.ccs.beans:

- se.ccs.beans
  - CCSCommandBean
    - Is used when you want run an application from the command line.
  - CCSDatabaseBean
    - This bean handles all the connections and queries to the database.
  - CCSEmailBean
    - For sending and reading email.
  - CCSFileBean
    - Creates deletes and download files.
  - CCSScheduleBean
    - Handles the scheduling mechanism.
  - CCSAddToRepositoryBean
    - Downloads files from a FTP.
  - CCSFileInfoBean
    - Applies metadata to content.
  - CCSImageMetadataBean
    - Extracts the EXIF tags from an image and convert them into metadata.
  - CCSInputDatabaseBean
    - Inserts the content with it’s metadata to the database.
  - CCSMetadataBean
    - Extracts metadata from content.
  - CCSMplayerInfoBean
    - Reads an audio or video-file and extracts its metadata.
  - CCSVASPBean
    - Sends content via a MUSIS channel.
  - CCSCConvertBean
    - Converts content into different formats.
Appendix D – GUI and Java applications used in the CCS system

We have also created a Java application for the scheduling mechanism:

- CCSScheduler
  - startNow()

To be able to test the functionalities of the CCS-system we implemented a GUI:

- CCS_CollectConvertSend
  - Different Java Server Faces (JSF) pages.

To get a file in the repository via TCP/IP, the following can be used:

- viewCCSTables
  - getFile.jsp
    - via HTTP, returns an item in the repository by letting the user provide a unique identification number that represents the file
Appendix E – Database structure used in the CCS system

The database used in the CCS system is designed to be flexible when it comes to storing metadata key and value pairs. It has a few basic tables that handle the relation between an object in the database and the associated metadata.

Table – ccs_repository

This table keeps a record of the object in the repository. When a new object is to be stored, one row in this table will be created and an automatic index number will be given in the field `repositoryID`. The repository ID can later be used to reference the object in various contexts, for example via the `getFile` interface described in Appendix D.

```
+--------------+------------------+------+-----+---------+----------------+
| Field        | Type             | Null | Key | Default | Extra          |
+--------------+------------------+------+-----+---------+----------------+
| repositoryID | int(16) unsigned |      | PRI | NULL    | auto_increment |
+--------------+------------------+------+-----+---------+----------------+
```

Table – ccs_metadata

The metadata table is a list of all available metadata that has been used so far. The table contains an identification number that automatically increments and the name of the metadata. Examples of metadata names could be `image height`, `name`, `light source`, `keywords`, `GPS latitude` etc.

Please note that the table only stores the metadata names, not the values. The values are stored and related to the repository ID in the table `ccs_rep2md`.

```
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| metadataID | int(16) unsigned |      | PRI | NULL    | auto_increment |
| metadata   | varchar(255)     |      |     |         |                |
+------------+------------------+------+-----+---------+----------------+
```

Table – ccs_rep2md

This table is responsible for the relations between the object and the metadata name, i.e. the repository ID and the metadata ID. If the table description below, the `ID` field is the ID where information about the object is. In our case, that is the table called `ccs_repository` and therefore we reference to it by naming the table in the `IDTable` field. By using this method, we can connect the metadata to something else than a repository object, for instance a table of users or some other entity that is not a repository object. We also store the timestamp of when the connection was made.

Please note that the metadata value is stored in the `value` field with the data type `text`, allowing anything that can fit into this data type to be stored. This is a drawback of the
generalization made in this database design. Normally you want an integer to be stored within an integer data type field, not a text field.

| Field      | Type             | Null | Key | Default           | Extra |
|------------+------------------+------+-----+-------------------+-------|
| metadataID | int(16) unsigned |      |     | 0                 |       |
| ID         | int(16) unsigned |      |     | 0                 |       |
| time       | timestamp        | YES  |     | CURRENT_TIMESTAMP |       |
| value      | text             |      |     |                   |       |
| IDTable    | varchar(255)     |      |     |                   |       |

**Database table description - ccs_rep2md**

**Table – ccs_namespace**

In this table we can store different namespaces that has been extracted from e.g. a JPEG image from mobile phone. When the JPEG metadata is analyzed, not only the `exif` namespace is used. For example, you can also find `canon makernote` when you use a canon digital camera or `GPS` if the image has GPS data attached to it. This depends on how the camera handles the metadata attachment in the image. The relation between the namespace and its metadata key- and value-pair is made in the table `ccs_rep2md`.

| Field       | Type             | Null | Key | Default | Extra          |
|-------------+------------------+------+-----+---------+----------------|
| namespaceID | int(16) unsigned |      | PRI | NULL    | auto_increment |
| URL         | text             |      |     |         |                |
| namespace   | varchar(255)     |      |     |         |                |

**Database table description - ccs_namespace**

**Table – ccs_ns2md**

The relation between a namespace and its metadata key- and value-pair is done using this table. It contains the relation by using the namespace ID, the metadata ID and the repository ID. Please note that the value can be found in the table `ccs_rep2md`, not here. A join of these tables is necessary in order to get the desired value.

| Field        | Type             | Null | Key | Default | Extra          |
|--------------+------------------+------+-----+---------+----------------|
| ns2mdID      | int(16) unsigned |      | PRI | NULL    | auto_increment |
| namespaceID  | int(16) unsigned |      |     | 0       |                |
| metadataID   | int(16) unsigned |      |     | 0       |                |
| repositoryID | int(16) unsigned |      |     | 0       |                |

**Database table description - ccs_ns2md**
Table – ccs_conversionMethod

This table describes what format that can be as input and output in the conversion part of CCS. The table also keeps track of what web service that is in charge of making the particular conversion. By using simple SQL queries, the system can decide on which conversions that can be carried out for a particular input format.

Please note that more information about the advantages of using this approach when it comes to plug-in and client web service generalization can be found in section 4.4

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>conversionMethodID</td>
<td>int(8) unsigned</td>
<td></td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>formatIn</td>
<td>varchar(128)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>formatOut</td>
<td>varchar(128)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSName</td>
<td>varchar(255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>displayName</td>
<td>varchar(255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database table description - ccs_conversionMethod

Table – ccs_eventScheduler

This table represents a simple scheduler that activates a certain event on a certain time. The table consists of two very important fields, the `eventName` field and the `parameters` field. The `eventName` is connected to some predefined logic in the code and `parameters` are a parameterization of the variables used in that event.

In Use Case 4 (see section 3.2.3), this table is used to store the variables concerning the URL of the RSS feed, which conversion to make, what e-mail account to send from and what e-mail address to send the converted content to.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventSchedulerID</td>
<td>int(16) unsigned</td>
<td></td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>time</td>
<td>varchar(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eventName</td>
<td>varchar(255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>mediumtext</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>monday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>tuesday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>wednesday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>thursday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>friday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>saturday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>sunday</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>daily</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>weekly</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>monthly</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>always</td>
<td>varchar(8)</td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
Database table description - ccs_emailAccount

Table – ccs_emailAccount

This table only purpose is to keep track of predefined e-mail accounts when scheduling content that is sent or collected via the automatic scheduler. In the table ccs_eventScheduler, an ID with a relation to a row in this table could be stored to keep sensitive data out of that table.

| Field          | Type             | Null | Key | Default | Extra          |
|----------------+------------------|------|-----+---------+----------------|
| emailAccountID | int(16) unsigned |      | PRI | NULL    | auto_increment |
| username       | varchar(255)     |      |     |         |                |
| password       | varchar(255)     |      |     |         |                |
| host           | varchar(255)     |      |     |         |                |
| protocol       | varchar(32)      |      |     |         |                |
| fullName       | varchar(255)     |      |     |         |                |
| emailaddress   | varchar(255)     |      |     |         |                |
| accountType    | varchar(32)      |      |     |         |                |
| port           | varchar(8)       |      |     |         |                |

Database table description - ccs_emailAccount
Appendix F - Source code for method aiff2amr in CCSConvertBean

public String[] aiff2amr(String inAiffURL) {
    // First rss2wave, then ffmpeg to amr
    String noAmrFilename = inAiffURL.replaceAll(".amr","\"\"\"\") ;
    String amrFileURL = noAmrFilename.replaceAll(".aiff",".amr");

    CCSCommandBean cmdBean = new CCSCommandBean();
    String aiff2amrCommand =
        "/Applications/ffmpegX.app/Contents/Resources/ffmpeg -i " + inAiffURL + " -ac 1 -ab 8 -ar 8000 -f amr -acodec amr_nb "+ amrFileURL;
    System.out.println("CMD: "+aiff2amrCommand);
    //String rss2amrCommand = "/usr/local/bin/ffmpeg -i " + waveFileUrl + " -acodec amr_nb -ar 16000 -ac 1 -ab 23 -y "+ amrFileURL;
    String commandResult = cmdBean.executeCmd(aiff2amrCommand);
    if (commandResult.compareTo("Command has been executed")==0) {
        return new java.lang.String[] {"Ok",amrFileURL};
    } else {
        return new java.lang.String[] {"Error in CCSConvertBean/aiff2amr: ", commandResult};
    }
}