A Context-aware Content Management and Presentation System for Pervasive Display Networks

Benjamin Hebgen

July 2009
## Table of Contents

Table of Contents ................................................................................................. 2  
Figure List ........................................................................................................... 5  
1. Abstract ........................................................................................................... 7  
2. Acknowledgement ........................................................................................... 8  
3. Problem Description ...................................................................................... 9  
   3.1. Introduction ............................................................................................... 9  
   3.2. State of the Art ......................................................................................... 11  
      3.2.1. Content Management ......................................................................... 11  
      3.2.2. Displaying Content ........................................................................... 11  
4. Problem Solution ............................................................................................. 12  
   4.1. Architecture .............................................................................................. 12  
   4.2. Overall Design ......................................................................................... 13  
   4.3. Component Design ................................................................................... 14  
      4.3.1. Output Controller ............................................................................... 14  
         Introduction .............................................................................................. 14  
         Background Information ........................................................................ 15  
         Context Management Framework .......................................................... 15  
         Ponder2 Toolkit ....................................................................................... 19  
         Architecture ........................................................................................... 20  
         Processing Unit for complex subscriptions ............................................ 22  
      4.3.2. Output Controller Client ................................................................. 23  
         Introduction .............................................................................................. 23  
         Architecture ........................................................................................... 25  
         Screen Manager ....................................................................................... 26  
         Detecting free rectangles on the screen .................................................. 26  
         Placing content ....................................................................................... 31  
      4.3.3. Communication Interface ................................................................. 32  
         Communication via the CMF .................................................................... 32  
         Self publishing of Output Controller Clients ........................................... 33  
         Self subscription of Output Controller Clients ....................................... 34
7. Bibliography ........................................................................................................................................59
8. Appendix ........................................................................................................................................62
  8.1. Supported file format list for Java Media Framework ................................................................. 62
  8.2. Supported file format list for FFmpeg ......................................................................................... 62
Figure List

Figure 1 architecture diagram ........................................................................................................ 12
Figure 2 use case diagram: Basic Use Case .................................................................................. 13
Figure 3 high level view Context Management Framework [5] .................................................... 15
Figure 4 network communications Context Management Framework [5] .................................... 16
Figure 5 Processing Unit anatomy [5] .......................................................................................... 18
Figure 6 class diagram: Output Controller ................................................................................... 20
Figure 7 class diagram: Processing Unit for complex subscriptions ........................................... 22
Figure 8 class diagram: Output Controller Client ......................................................................... 25
Figure 9 free space algorithm part 1, step 1 .................................................................................. 26
Figure 10 free space algorithm part 1, step 2 ................................................................................ 27
Figure 11 free space algorithm part 1, step 3 ................................................................................ 27
Figure 12 free space algorithm part 2, step 1 ................................................................................ 27
Figure 13 free space algorithm part 2, step 2 ................................................................................ 28
Figure 14 free space algorithm part 2, step 3 ................................................................................ 28
Figure 15 free space algorithm problem ...................................................................................... 29
Figure 16 extended free space algorithm step 1 .......................................................................... 29
Figure 17 extended free space algorithm step 2 .......................................................................... 30
Figure 18 Output Controller Client entity .................................................................................... 33
Figure 19 sequence diagram: load policies ................................................................................... 37
Figure 20 sequence diagram: setup processing units and policy activation .................................. 38
Figure 21 sequence diagram: execution of a policy ...................................................................... 39
Figure 22 high level view FOBS .................................................................................................. 41
Figure 23 layer design .................................................................................................................. 49
Figure 24 sequence diagram: add new content to screen ............................................................ 50
Figure 25 sequence diagram: manage screen .......................................................... 51

Figure 26 Screenshot Output Controller Client Demo. Default commercial. ....................... 54

Figure 28 Screenshot Output Controller Client Demo. Displaying Bluetooth services and a calendar...... 55

Figure 29 Screenshot Output Controller Client Demo. Displaying two calendars. ...................... 55
1. Abstract

This thesis describes a way of managing and displaying content which is based on context information originating at the Context Management Framework from NEC.

In order to automate the management of the content data, a policy based system using event-condition-action policies has been designed and implemented.

The main application of this system is in the digital signage area, where large displays with multiple observers are common. Each requires different information and wants it shown as quickly as possible. Therefore a set of metadata has been defined which describes which kind of content should be displayed and in what manner.

In order to make the most out of large displays, a presentation system has been designed and implemented which can dynamically manage, display, position and resize several contents on one screen, based on the metadata. This system is capable of displaying the most common media types.

As explained in the demonstration section, the system has been implemented showing the ability to manage the content of several displays to show personalized information to multiple users, based on the context in which they appear.
2. Acknowledgement

I would like to thank everybody who helped me finishing my studies and this thesis. Especially I thank my parents for supporting me all the way during my studies and this thesis.

I would like to thank my three supervisors at NEC Miquel Martin, Martin Bauer and Martin Strohbach for all the support they gave me and the discussions we had which always helped me advancing in the thesis.

I also like to thank my supervisor at the Hochschule Mannheim Prof. Dr. Miriam Föller-Nord for her support and assistance during my thesis.

I like to thank Dr. Kevin Twidle for the time he spent in conversions with me helping me to understand and use the Ponder2 system.
3. Problem Description

3.1. Introduction

As part of the digital signage trends, traditional devices for public information like advertising columns or information displays (e.g. with flight schedules), are being replaced by modern flat screen displays, whose content is delivered over the network, or through the use of customized set-top boxes.

Since these set-top boxes are getting more and more technical advanced there are more opportunities to use the displays connected to them. Instead of just showing static or time scheduled information these displays could be used to show dynamic, personalized information.

In order to determinate the most relevant information for an audience, and to personalize it accordingly, it is required to identify those persons which are standing in front of such a display and the context of their activities. In other words, there is a clear need for context-aware content.

In Digital Signage, one can change the contents of a display at the touch of a button, this saving the work of tearing off the old paper poster. The possibilities, however, are far beyond such simple operations. By using context information, a much higher level of interactivity and personalization can be reached; a wide range of technologies can be used to increase interaction and personalization. As an example, face recognition can be used to identify a person, and waving an accelerometer-equipped mobile device can be interpreted as navigation cues.

Hence large displays are common in digital signage and its most common environment is public places [2] with relatively large audiences (e.g. passer-bys). Such displays could then be used to simultaneously display several personalized contents based on the perceived context information.

Although the main focus of this thesis is on digital signage there are a more use cases where context-aware content is required.
One such example is assisted living, where context based information can be used to interact with certain devices. From a simple “switch the light on when someone enters the house” over gesture control of devices to complex medical support use cases, there are a lot possibilities where contextual information could be used.

This thesis will describe the design and implementation of a content management and display system which manages the content based on context information. Therefore a server and client system has been designed and implemented consisting of

- A server named the Output Controller. A system which provides content information based on context information.
- A number of clients referred to as Output Controller Clients. A system which is capable of rendering and displaying several contents on one screen based on the provided information.

Sensor data is provided by NEC’s Context Management Framework (CMF), which has been extended and integrated.

The platform of choice for the implementation of this system is Java. The Context Management Framework is also developed in this language, for the same reasons that apply to this case: the code is highly portable, making the deployment in heterogeneous platforms much easier (e.g. mobile phones, desktop PCs, Set-Top Boxes using either Linux, MacOS or Windows).

To achieve the desired level of automated content management, the Output Controller is designed around the principles of a PDP/PEP (Policy Decision/Enforcement Point). This thesis will give an overview over the used policy framework Ponder2 and show how the interaction between the Output Controller and Ponder2 is established.

Additionally this thesis will describe a way of combining information from various data sources connected to the CMF. Furthermore this thesis will describe a set of metadata used to command the Output Controller Client. This metadata will cover the needed information to display content properly on the right location on a display.

The description of the Output Controller Client will show a way to manage several contents on one screen. Therefore an algorithm which detects the available free rectangles has been developed and will be described. Additionally it will be shown how clear separate layers can be provided using Java6.

Hence the Output Controller Client is designed to show all common media types, an overview over Java’s possibilities to handle media output and a summary of the made research on this topic will be shown.

In order to give a good overview over the implementation the most common scenarios will described.

As final part a demonstration will be shown followed by a conclusion and an outlook discussing the available working areas for this system and how modifications could extends the range of working areas.
3.2. State of the Art

3.2.1. Content Management
A state of the art analysis showed that there is for now no context based content management system available.
There are several content management systems available in the digital signage area. They are mostly time scheduled but none of them provides a generic way to manage content based on context.
Some solutions include context based content to certain limit like a CD-preview on a terminal based on a scanned barcode or interactivity with touch screens which can be seen as context. But these are specialized and isolated solutions. [1; 2]

In assisted living the focus is most times very use case specific [3]. There are systems available which take advantage of sensor technology or other context information sources. For instance a movement detector gets activated and the light switches on or a camera detects missing food in the refrigerator. But these systems are also seperated systems which only work isolated and have a specialized goal.

3.2.2. Displaying Content
A state of the art analysis showed that there is no system available which is capable of dynamically managing several contents on one screen.

In the digital signage area there are a lot of displaying systems available which are capable of displaying various types of media. But since the whole content management in the digital signage area is mostly based on time schedule these systems are designed to show one content at one time.

The systems which are capable of displaying various contents at one time are used case specialized systems, like a display showing several flight plans at one time. These systems are static designed with exact predefined areas where which content is shown.
4. Problem Solution

4.1. Architecture

The Architecture of the system is based on a three tier architecture where the CMF basically represents the data tier, the Output Controller represents the logic tier and the Output Controller Client represents the presentation tier.

Within the CMF the Processing Units have a special position. Hence they do provide data but also work on data they received they are part of the data tier and the logic tier.

The point where the system differs from the classic strict linear three tier architecture is the communication between the components. Hence the CMF is used for communication between Output Controller and Output Controller Client there is a direct communication between Presentation and Data Tier.
4.2. **Overall Design**

The main concept of this system consists of three components.

- The Context Management Framework
- The Output Controller
- One or more Output Controller Client.

In the basic use case the CMF gets an input from an attached data source, in most cases a sensor. Trigger by this the CMF sends a notification to the Output Controller which has subscribed to this context information.

The next step the Output Controller queries the CMF for information about the content which should be shown according to the context. This content data is sent to an Output Controller Client.

An Output Controller Client renders the content. The rendering can be done internally by the Output Controller Client itself or an external renderer can be used.

Furthermore the Output Controller Clients register them-selves at the CMF in order to receive information from the Output Controller.

![Figure 2 use case diagram: Basic Use Case](image)
4.3. Component Design

4.3.1. Output Controller

Introduction
The Output Controller was created to provide an automated way to manage content on various displays based on context information provided by the CMF.

In order to achieve the automation of the content management the Output Controller was designed as a policy based system using the Ponder2 Toolkit\(^1\). Therefore a communication between the Output Controller and Ponder2 was designed.

For the communication between to Output Controller Clients a communication interface was implemented. The communication interface uses the CMF as communication bus\(^2\) and a set of metadata\(^3\) to provide content information to the Output Controller Clients.

The Output Controller is capable of combining several subscriptions to the CMF to provide a higher level of personalization for the provided content.

Hence the CMF doesn’t provide a way to logically combine subscriptions a Processing Unit was designed providing a simple AND of OR logic to combine subscriptions.

To setup the Processing Units and receiving the actual context information from them the Subscriber class was created.

Additionally the Inquirer class was created to provide extra information from the CMF if required.

---

\(^1\) See chapter Ponder2 Toolkit at page 31
\(^2\) See chapter Communication via the CMF at page 24
\(^3\) See chapter Meta Data at page 25
Background Information

Context Management Framework

Overview
The CMF is a framework for collecting, analyzing and utilizing context. The architecture is designed to meet the requirements of global, enterprise, and Personal Network scenarios.

On the high level view the CMF [4] basically consists of 5 parts.
1. The Data Source Abstraction Layer (DSAL) which has wrappers for the actual sensor and puts the sensor data into a unified data structure which is send to the Context Agent Core
2. The Processing Units are used to provide dynamic data to the Context Agent Core which is either based on the operating system (e.g. a timer) or on sensor data or a combination of both.
3. The Storage Component is used to provide data which doesn’t change so frequently (e.g. from a database).
4. The Context Agent Core collects all the data and provides them to the Application Interface. Furthermore it can retrieve input data from the Application Interface and put the data into the suitable subcomponent.
5. The Application Interface provides a XML based language, named Context Access Language (CALA). With this language a user is capable to subscribe, to query, to update and to insert data to the CMF or to delete data from the CMF.

Figure 3 high level view Context Management Framework [5]
Networking
On the Network part the CMF works as a distributed application where every node is capable of forwarding requests into the network.

![Diagram showing network communications Context Management Framework](image)

**Figure 4 network communications Context Management Framework [5]**

Context Model
The CMF uses an entity data structure to represent data which is provided by the Context Agent Core. The data structure is represented in CALA. In this data structure every entity has a type and within the type a unique identifier. In addition an entity can have one or more attributes. These attributes have a name and a value. The name of an attribute does not have to be unique. This means multi value attributes are possible by having various attributes with the same name.
**Context Access Language (CALA)**

CALA is a XML based query language which is provided by the Application Interface. CALA supports the three different types of interactions:

- Synchronous retrieval: query/response
- Asynchronous retrieval: subscription/notification
- Synchronous modification: insert/update/delete

All interaction types have the following elements:

- A selector which allows the selection of an entity either based on the identifier or based on the type of the entity. Within the Selector there are also the attributes on which the interaction will take place
- A scope which limits the range of the interaction within the CMF to either the local CMF node, the current CMF cluster or the entire CMF network
- A restriction which allows a more detailed selection based on the value of an attribute, similar to the WHERE clause in SQL
- An option where additional requirements like quality level for the selected context can be defined.

In addition subscriptions have a subscription condition which sets the attribute which should be watched by subscription. If the value of this attribute changes the notification is fired. The CMF provides 2 communication interfaces to execute CALA commands. A XML Remote Procedure Call interface and a set of Java classes which represent the CALA components and Java class as client which allows submitting the CALA command to the CMF.
**Processing Units**

The anatomy of a Processing Unit consists of four parts.

The two core parts of every Processing Unit

- The basic Processing Unit which contains all necessary functionality to interact with the Processing Manager
- The Processing Manager which is responsible to manage all requests coming out of or into a Processing Unit.

And two customizable parts

- The Processing component which extends the Processing Unit and is used to modify the data provided by the Processing Manager
- A mapper turns the raw data into CMF entities. If it is required to model the entities in more than one way, one can create multiple mappers for the same source

![Figure 5 Processing Unit anatomy [5]](image-url)
Ponder2 Toolkit

“Ponder2 comprises a self-contained, stand-alone, general-purpose object management system with message passing between objects. It incorporates an awareness of events and policies and implements a policy execution framework. It has a high-level configuration and control language called Ponder Talk and user-extensible managed objects are programmed in Java.” [6]

Ponder2 provides an event case action policy system which is used for the Output Controller. Using the Ponder Talk language Ponder2 can be used to control Java classes.

Ponder Talk

Ponder Talk is a high level language which is based on Smalltalk. Ponder Talk is used to control and interact with Ponder2. Ponder2 provides three interfaces for Ponder Talk

- P2 files which are read in and executed. This can be done during the start by a startup parameter or directly using Ponder Talk
- A Telnet connection which provides a Ponder Talk console interface
- A set of Java classes which allow execution of Ponder Talk commands out of Java
Architecture

The Output Controller can be divided into three sections.

- The communication with the CMF
- The communication with the Output Controller Clients
- The communication with Ponder2
The CMF communication section basically consists of the Inquirer, the Subscriber and the SubscriberContainer classes. The Inquirer is used to combine various queries. Therefore it provides a function set to store the needed queries before sending them to the CMF and merging their results into one result.

The Subscriber is used to combine various subscriptions in a logical AND of OR list. Therefore the Subscriber holds an AndList object and provides a function to generate a new OrList object and add this object to the AndList object. The OrList provides a function set to add a new Subscription. The result of combined subscriptions is not obvious and has to be defined. Therefore the AndList class provides a function to define the result. This result can either be the result of one OrList or a combination of several OrList results. The result of an OrList is the result of the subscriptions which are active.

The communication with the Output Controller Clients is managed by the OutputManager class. Therefore the OutputManager class provides various functions to define the content and its metadata. Furthermore various functions are provided to select the content data receiving Output Controller Clients by their attributes.

The communication with Ponder2 is achieved by two separate techniques. To send commands to Ponder2 a function from the Ponder2Connector class is used. Receiving commands from Ponder2 is accomplished by assigning a Java annotation to a function\(^4\). In order to make the Output Controller controllable by Ponder2 these annotations are spread all over the whole Output Controller.

\(^4\) See chapter Communication Ponder2 to Java at page 34
**Processing Unit for complex subscriptions**

To compensate the missing ability of CALA to combine several subscriptions a flexible Processing Unit is used to provide simple operations to combine subscriptions.

Therefore this Processing Unit has the SubscriptionOrList class and the SubscriptionAndList class to provide a simple AND of OR logic.

The SubscriptionOrList class has an array of Subscriptions. These Subscriptions are combined by or. This means if one of them is activated the SubscriptionOrList is activated. The result of the SubscriptionOrList is the result of the subscription that is last activated.

The SubscriptionAndList class has an array of SubscriptionOrList objects which are combined by AND. This means all of them have to be active to activate the SubscriptionAndList.

Hence the result of combined subscriptions is not obvious and has to be defined the ResultMapper class is contained in this Processing Unit. The ResultMapper is a configurable mapper which can either be parameterized to generate a static predefined result (e.g. a string) or to generate a dynamic result which is the result of a SubscriptionOrList.

In addition to that the normal subscription is extended with a time out. The combination of stateless sensors with a time out leads to the simulation of a stateful sensor. There are various stateless sensors (e.g. RFID) which need the time out. Stateful sensors are needed for logic operations because a stateless sensor has a value just for the moment it’s activated which makes it basically not possible to perform a logical and operation with this value at any point in time other than the event occurrence.

![Figure 7 class diagram: Processing Unit for complex subscriptions](image-url)
4.3.2. Output Controller Client

Introduction
The main goal for the Output Controller Client is to receive Meta data about some content and then display the content based on the Meta data. The metadata\(^5\) is defined by the communication interface between Output Controller and Output Controller Client.

In order to display several contents at the same time the ScreenManager was created to manage adding, removing, positioning and resizing of content on multiple layers.

The Output Controller Client is designed to be able to display the most common formats for describing information on a Swing GUI\(^6\). The list of formats that are supported by this application is:

- Most common video formats
  - DivX
  - Xvid
  - Windows Media Video
  - Mpeg 2
  - Mpeg 4
  - Quicktime Movie

- Most common image formats
  - Bmp
  - Jpeg
  - Gif
  - PNG
  - SVG

- Most common document formats
  - PDF
  - HTML

[^5]: See chapter Meta Data at page 26).
[^6]: For more information about Swing see [37]
- Plain Text
- Office Documents
  - Microsoft Word
  - Microsoft Excel
  - Open Document Text
  - Open Document Spreadsheet
**Architecture**

The OutputControllerClient basically consists of three parts: the ScreenManager, a set of different content type classes which implement the common interface MediaContent and their visual components which extend JPanel.

The ScreenManager holds the main window of the OutputControllerClient and has various functions to manage the content.

Each content class has to implement various functions to control its visual component and to make the visual component accessible to the ScreenManager.

The content classes represent basically the primary MIME types. Their visual components are capable to display all the needed MIME subtypes.
**Screen Manager**

In a first approach to manage several contents on one screen it was considered to use a solution based on a modified version of the rectangle packing problem. The rectangle packing problem is the problem of packing several rectangles of different size into one big rectangle using as much space as possible of the big rectangle\(^7\).

The main problem here is that the rectangle packing problem assumes a free movement of all rectangles which is not possible in the given problem. In order to provide a good usability to the end user it is necessary to keep a displayed content in the area where it first appears. A content which appears on a different random position on the screen every time a new content is added to the screen makes the provided information basically useless for the end user.

Aside from that the rectangle packing problem doesn’t take any account of metadata like a priority.

Therefore the idea was dropped and an algorithm was created which detects free spaces on the screen and tries to display new content in proper way based on the available free space and the provided metadata.

**Detecting free rectangles on the screen**

Hence an algorithm which checks every pixel for its usage is taking a lot of time\(^8\) and the running time is growing with the screen size a solution is needed where the running time is not connected to the screen size and where the running time is within a timeframe of maximal one second.

The algorithm to detect the free rectangles is based on having all size and location information about all contents that are displayed.

The main idea of this algorithm is to get the biggest vertical and horizontal rectangles that are creatable from two diagonally opposite corners of all displayed contents. For the actual implementation of the algorithm and the use case tests the upper left and the lower right corner are used.

In order to detect the biggest vertical rectangle that is creatable from a corner a line, which starts from one pixel below or above the corner in free space and ends when it hits content or the screen border, is taken.

\[\text{Figure 9 free space algorithm part 1, step 1}\]

\(^7\) For additional information see
\(^8\) See chapter Testing and Profiling at page 52)
In the next step it is detected how much this line is horizontal moveable till it hits content or the screen border.

Finally the location and the size of the rectangle are calculated. The location is calculated by taking the start of the line as Y-coordinate and the maximal movement of the line to the left as X-coordinate. For the size the height is calculated by the difference between the start and the end of the line and the width is calculated by the difference between the maximal movement of the line to the left and to the right.

The way to detect the biggest horizontal rectangle is almost the same as for the vertical rectangle.

Instead of a vertical line a horizontal line, this starts from one pixel left or right from the corner in free space and ends when it hits content or the screen border, is taken.
In the next step it is detected how much this line is vertical moveable till it hits content or the screen border.

![Figure 13 free space algorithm part 2, step 2](image1)

For the calculation of the location the start of the line is taken as X-coordinate and the maximal upward movement of the line is taken as Y-coordinate. The height is calculated by the difference between the maximal upward and downward movement of the line. The width is the difference between the start and the end of the line.

![Figure 14 free space algorithm part 2, step 3](image2)
After a few tests there have been several use cases where the algorithm failed to detect all available rectangles.

![Image of detected and missed rectangles](image)

**Figure 15 free space algorithm problem**

By taking two possible maximal vertical and horizontal rectangles of the two corners the whole free space on the screen is covered and all possible maximum rectangles are detected.

After this slight modification of the algorithm it succeeded in every tested use case.

In order to get the second vertical rectangle the line for the algorithm is created by starting one pixel to the left or the right and setting the start point of the line to the highest point reachable without interference by a content object.

![Image of extended free space algorithm step 1](image)

**Figure 16 extended free space algorithm step 1**
The second horizontal rectangle is detected by almost the same way. The line for the algorithm is created by starting one pixel to upward or downward and setting the start point of the line to the most left point reachable without interference by a content.

Hence this algorithm detects some rectangles multiple times those have to be sorted out to avoid duplicates. In the worth case a rectangle is detected two times the number of contents on the screen.
Placing content

The general idea for the algorithm to place content on the screen is to find the best fitting for the new content. Hence content can have a location hint the best fit is the location closest to the location hint. If the content has no location hint the best fit is the free rectangle which has the least overhead of free space.

With location hint

In order to get the closest possible location to the location hint all free rectangles are checked if they contain the wanted location or whether they are on the left or the right side of the wanted location and whether they are above or below the wanted location.

If a free rectangle contains the wanted location and the content fits in on that location the algorithm can stop because the best fit is found.

If the content doesn’t fit in the rectangle on the wanted location, the closest possible location to the wanted location inside the rectangle is taken and the distance between the two locations is calculated by the Pythagorean Theorem.

If the content doesn’t fit in any of the free rectangles on the wanted location the rectangle which can fit in the content on the location with the least distance to the wanted location is the best fit.

Without location hint

In case of content without a location hint all free rectangles are checked whether the content fits in or not. If the content fits in the free rectangle the overhead of free space is calculated by a simple division of the two areas.

\[
\text{overhead} = (\text{free rectangle width} \times \text{free rectangle height}) - (\text{content width} \times \text{content height})
\]

The free rectangle with least overhead is the best fit for the content.
4.3.3. Communication Interface

*Communication via the CMF*

Hence the CMF can be used as communication bus by its subscription and notification services it was decided within the project group to use the CMF as communication interface between an Output Controller Client and the Output Controller. This decision was made because of two reasons. The main reason was to extend the working area of the CMF from providing information to generating triggers based on the information.

The second reason was to avoid unnecessary usage of additional technology which would require additional research and additional computing power.
Self publishing of Output Controller Clients

In order to achieve the communication between the Output Controller and the Output Controller Clients the Output Controller Clients publish them self to the CMF at their start. Therefore each Output Controller Client creates a new entity of the type output. The Output Controller uses this entities to identify the running Output Controller Clients and to add content to them or remove content from them.

![Diagram of Output Controller Client entity]

Figure 18 Output Controller Client entity
**Self subscription of Output Controller Clients**

Each Output Controller Client subscribes to the addContent and the removeContent attribute of its entity. If one of the entries changes the Output Controller Client adds new content or removes content based on its id.

**Meta Data**

In order to display a new content properly the submitted metadata consists of the following

- An ID which is unique within the system.
- A source which represents the location of the content to be displayed.
- A MIME type which is used by the Output Controller Client to determine the correct way to display the content.
- A priority which is used to place the content in the content queue of the Output Controller Client.
- A stop type which tells the Output Controller Client if the content is removed by an Output Controller call or by a time out or at the end of the content if its playable media like audio or video files.
- A time out which sets the time how long a content should be displayed if the time out stop type is set.
- The layer on which the content will be displayed.
- A location hint which defines a preferred place of appearance on the Output Controller Client. This location hint is optional.
4.4. Implementation

4.4.1. Output Controller

Technical Background

Communication Java to Ponder2
In order to communicate with Ponder2 out of Java Ponder2 provides the P2Compiler class and the 
XMLParser class. The P2Compiler is used to convert a Ponder Talk command into an internal used XML 
format and the XMLParser is used to execute the XML command.

Communication Ponder2 to Java
In order to access the functionality of a Java class Ponder2 provides the ManagedObject interface which 
signals Ponder2 to generate adaptor for the communication between Ponder Talk and the implementing 
class. By implementing the interface into a Java class an object factory can be created using Ponder Talk. [7]

```
root/factory at: "subscriber" put: (root load:
"de.nec.nle.outputcontroller.cmfcommunication.Subscriber").
```

For the actual access to the Java functions Ponder2 uses Java annotation (@Ponder2Op("keyword")). 
These annotations are used to set the Ponder Talk keyword which is used to call the underlying function. 

As a minimum prerequisite to create a new object using Ponder Talk, the constructor of the Java class 
has to be marked by a Ponder2 annotation with the keyword “create”.

```
@Ponder2op("create")
public Subscriber() {
    ...
}
```

To access a normal Java function the keyword can be chosen randomly but has to be unique within the 
class.

```
@Ponder2op("eventName")
public String getEvent() {
    return this.event;
}
```

In case the Java function provides a return value it is passed through to Ponder2. In order to access a 
parameterized function a colon has to be added to the keyword. Functions with multiple parameters can 
be accessed by attaching the corresponding number of keywords within the annotation.

```
@Ponder2op("addResult:value:")
public void addResult(String name, String value) {
```
Return values and parameters Ponder2 can handle, have to be either basic data types or have to derive
from P2Object. Ponder2 provides a set of P2Object derived classes which basically represent the basic
data types of Java and in addition an array and a hash map data structure.

**Evaluating policies**
Ponder2 has two types of policies, Authorization Policies and Obligation Policies, also known as Event
Condition Action (ECA) policies. For this project the ECA policies are used. [8]
Ponder2 provides a factory to create a policy object.

```plaintext
policy := root/factory/ecapolicy create.
```
The empty policy now has to be filled with the event, the condition and the action information.

An event object can be created from the event factory.

```plaintext
event := root/factory/event create: #( "index1" "index2" ).
```
In Ponder2 each event is connected to an array. As soon as an entry is written into this array the event
will be activated. The values written into the array can be accessed inside the policy.

The event has to be attached to the policy.

```plaintext
policy event: event.
```
Hence the event for a policy is based on a subscription the condition part of the policies is not used.
Instead of that the condition for activation is included in the subscription by its restriction.

The action part of the event is a Ponder Talk block.

```plaintext
policy action: root print: "POLICY HAS BEEN ACTIVATED".
```
As last part the policy needs to be activated.

```plaintext
policy active: true.
```
**Scenarios**

**Load policies**

During the startup of the Output Controller the policies which are saved in P2 files are read in by the PolicyImporter class and sent to Ponder2 for execution. As part of the execution a Subscriber object is created. After the creation this object is added to a container. Depending on the content of the P2 file new OrList objects are created, added to the subscriber and filled up with subscriptions. Furthermore the result value and the result type of the subscriber are configured.

As final part of the setup the actual Ponder2 policy and the corresponding event are defined. Therefore the Subscriber object provides a unique event name.

![Figure 19 sequence diagram: load policies](image-url)
Setup Processing Units and policy activation

After all P2 files are read in and executed the container for the Subscriber objects serializes them into XML files. In the next step the container writes the configuration file for the Processing Units and starts the CMF system. Controlled by the configuration file the CMF instantiates the Processing Units and their mappers and provides them the corresponding serialized Subscriber object. After the de-serialization the Processing Unit starts the subscriptions stored in the Subscriber object.

The CMF system notifies the Processing Unit if the subscribed data changes. If the logical combination of the subscriptions within the Processing Unit is fulfilled the Processing Unit updates the data of the CMF. The CMF finally gets the formatted result from the mapper and notifies the Subscriber object which is subscribing to the Processing Unit. After the notification a Ponder2 event is triggered and the corresponding Ponder2 policy gets activated.

![Sequence Diagram](image)

**Figure 20 sequence diagram: setup processing units and policy activation**
Execution of a policy

After a policy gets activated Ponder2 sets up an OutputManager object which is responsible for spreading the information about new content which should be displayed. Therefore the OutputManager object first queries information about the available Output Controller Clients from the CMF. After that the actual content which should be displayed is defined. Finally Ponder2 tells the OutputManager object to add the new content to the Output Controller Clients. In order to do this the OutputManager object first determines meta information, like the MIME type, about the new content. Afterwards the new content information gets published through the CMF. In order to perform this publication each Output Controller Client has to create an entry for its self in the CMF system and subscribe to this entry.

Figure 21 sequence diagram: execution of a policy
4.4.2. Output Controller Client

Technical Background & Research

Video output with Java
Video output is one of the weak points of Java. While the Java core libraries don’t give any support to play videos on a Swing or AWT GUI [8] there are several approaches to get rid of this gap. Some of them are platform depended (e.g. a Direct3D wrapper). Since this is a no go for a platform independent design I’m only going to detail the platform independent approaches.

Basic Java Media Framework (JMF)
Sun’s first approach to close the video or the overall dynamic media gap in Java was the Java Media Framework\(^9\). The JMF offers the developer an easy way to load a media file, instantiate a player for the file and get an AWT or Swing component with the video laying on it. Since the JMF is developed by Sun and is available since a few years the documentation is quite good [10]. The biggest problem with the meanwhile aged JMF is the lack of supported media formats (Error! Reference source not found.). With the lack of a MPEG-2 and MPEG-4 Video support most modern formats like DivX, XVID or DVDs are not playable with the basic JMF.

Java FX
Java FX is Sun’s new scripting language for rich internet applications and has a strong focus on displaying media. Java FX can interact with Java, it can be embedded in a Swing application and the runtime is included in Java 6 but it is a complete new language [9].

Aside from the fact that Java FX is mainly for web applications there are various unsolved problems with it.

Java FX is not platform independent. There is no release for Linux [10]. On MacOS playing videos is basically not possible and played videos on Windows often had pixel errors which resulted in bad video.\(^{10}\)

Freedom for Media in Java (FMJ)
FMJ is basically an FFmpeg wrapper for Java. The goal of FMJ is to be API compatible with the JMF [11]. Although the developers of FMJ claim that there are no additional installations to the JRE needed it is not true. Since FMJ is still under development there are still parts which are not implemented yet and where the original JMF components have to be used. Aside from that the main problem with FMJ is that FMJ after several tries was not able to play videos with FFmpeg on Windows. So platform independency is not provided by FMJ.

---

\(^9\) For more information visit http://java.sun.com/javase/technologies/desktop/media/jmf/

\(^{10}\) Tested release of JavaFX was 1.0 with the release of JavaFX 1.2 a Linux release is available
**VideoLAN VLC Java Bindings (JVLC)**

VLC is supporting almost every actual media format [12] and so does JVLC [13]. It is nice and easy to use but it has 4 problems which prevent the usage of it.

The main problem with JVLC is its stability. While resizing or moving the visual component of JVLC it happens that the visual part of the video disappears while the sound keeps playing.

The second problem is that JVLC only provides an AWT component [14]. Since mixing up Swing and AWT components can produce unwanted behavior of a GUI [15] JVLC should only be used very carefully in a Swing GUI.

Maybe the biggest problem for developers is the lack of a usable documentation. There is a Javadoc available for it but there are absolutely no comments in it [14].

Another problem with JVLC aside from technical points is the license of it. Since it’s under GPL which is very restrictive makes it basically impossible to use it as a company to produce a product which should be sold.

**JMF with Fobs Plug-In**

FOBS is an object oriented wrapper for the FFmpeg library11. The goal of FOBS is to provide an easy to use interface between the ffmpeg C library and common high level languages like Java, C++ or Python.

![Figure 22 high level view FOBS](http://fobs.sourceforge.net)

On the Java side of FOBS the interface is provided as a plug-in for the JMF, called FOBS4JMF. The big advantage of FOBS4JMF compared to other approaches is that the developer can totally rely on the JMF and its good documentation.

---

11 For more information about ffmpeg [http://ffmpeg.mplayerhq.hu/](http://ffmpeg.mplayerhq.hu/)
FOBS4JMF provides all the ffmpeg supported formats (see table) to the JMF. In addition to the wide range of supported formats ffmpeg is capable of using codecs that are natively provided by the operating system.

Hence FFmpeg is platform independent [16] the same applies to FOBS.

Discussion
The fact that the basic JMF doesn’t support most of the common video formats makes it pretty unusable for this application.

Java FX is a brand new technology by Sun which is supposed to be a competitor to Adobe’s Flash technology. It has some nice ideas and features like using native codecs from the operating system but hence it doesn’t support platform independency and the video quality is quite poor yet it’s not usable for this application. If Sun improves the video quality and adds additional platform support in future Java FX should be considered as a possibility to play video files in Java.

Since it was not possible to get FMJ running under Windows a judgment about the supported formats and the video quality isn’t possible. But the fact that it doesn’t work under Windows makes it unusable for this application.

Aside from the fact that JVLC isn’t usable because of its license it has some major technical problems, where the stability is the worth one, and a total lack of documentation which makes it complete unusable.

The combination of JMF and FOBS offers the developer the easy usage and the good documentation of the JMF combined with a large number of supported formats, the possibility to use native codecs and a good video quality. Overall JMF with FOBS is the best solution to play videos with Java.
Image output with Java

Bitmap images
Hence the standard Java Image IO API is capable of handling every common bitmap image format [17](e.g. bmp, jpeg, png, gif) there are no additional libraries needed for handling bitmap images. In the Output Controller Client images read in are painted on a JPanel.

Vector images
Java has no build in support for vector images. Hence the Output Controller Client should be capable of the most common media formats this part is focused on displaying SVG graphics.

For Java there are basically two SVG libraries available, the Apache Batik Toolkit and the SVGSalamander project.

Apache Batik Toolkit
Batik is a complete toolkit for SVGs. It supports reading and writing of a SVG file, modifying and rendering. Batik supports almost the complete SVG 1.1 W3C Recommendation [18]. Hence Batik uses some additional libraries, like its own DOM and XML libraries, its size with around 5 MB is quite big.

The time to load and render a SVG file isn't really good.

SVGSalamander
SVGSalamander is a small SVG renderer package and is focusing on fast loading speed. The main intention of SVGSalamander is to give an easy to integrate SVG images into Java based videogames.

Although the SVGSalamander was developed for videogames does not make useless. The main feature that is needed for videogames, fast loading speed, matches perfect with the requirements for this system.

Discussion
Batik has a lot of advantages regarding the supported SVG functions but regarding loading speed and size SVGSalamander has the clear advantage. In addition to that is the usage of SVGSalamander a lot easier.

Hence for this application loading speed was important SVGSalamander is used. But the lack of supported SVG features, the stopped development and the fact that its successor can’t be used because of license problems makes it necessary to find a suitable replacement in future.
Document output with Java

Plain Text
Plain text can be directly displayed using various standard Swing components. For this application the JLabel is used.

HTML
In order to display a HTML document in Java there are two solutions to achieve this.

1. A HTML renderer for Java which actually displays the document with Java components
2. Embedding an external web browser into the GUI.

The main problem with the available HTML renderers is the lack of common plug-ins for them. Even if some of them actually support CSS and basic JavaScript that is not enough in a world of Web2.0 where enhanced technologies like Flash animations, AJAX or ActiveX are common.

Embedding a web browser into a Swing application can be achieved by two ways [19]. One way which is Windows only and therefore not going to be detailed is to use one of the available OLE libraries (Object Linking and Embedding) which allow to embedded OLE based applications, like the Internet Explorer, into Java.

The alternative is to use the XPCOM bridge (Cross Platform Component Object Model) from Mozilla [20] which can be used to embed a Gecko\(^{12}\) based browser. As of today there are 3 libraries available which use the XPCOM. The Swinglabs Java Desktop Integration Components\(^{13}\), the JRex project\(^{14}\) and the JxBrowser\(^{15}\) which is commercial and therefore not detailed.

Java Desktop Integration Components (JDIC)
The main goal of JDIC is to integrate native desktop applications into Java without losing platform independency [21]. As part of this goal JDIC offers an easy to use web browser component which renders the requested document on a Swing component. Therefore JDIC is using a local installed web browser. This web browser has to be capable of XPCOM. For now there is only support for the Mozilla browser. Firefox is not supported [22].

On Windows JDIC does also support embedding the Internet Explorer via OLE [22].

---

\(^{12}\) Gecko Engine: A web browser engine developed by Mozilla.
\(^{13}\) For more information visit [https://jdic.dev.java.net/](https://jdic.dev.java.net/)
\(^{14}\) For more information visit [http://jrex.mozdev.org](http://jrex.mozdev.org)
\(^{15}\) For more information visit [http://www.teamdev.com/jxbrowser](http://www.teamdev.com/jxbrowser)
**JReX**

The idea of JReX is to provide a browser component for Java. Therefore JReX uses a built-in Gecko based browser called GRE [23]. This means there is no need for an additional installed browser. The Gecko base allows JReX to use Mozilla plug-ins [24].

JReX offers a complete browser window, including all control elements. This browser window can be configured to have tabs or just a single window but there is no component available which only displays the document [24].

**Discussion**

While both solutions are working quite well JDIC has the clear advantage in case of usability for the Output Controller Client. Although JReX doesn’t need a preinstalled web browser, the fact that JDIC provides a Swing component which displays only the rendered document without any decorations or control elements makes it the best choice.

---

**PDF**

While there are quite a few PDF parsers available on the internet there are only two PDF libraries available which support rendering. Sun’s Swinglabs PDF Renderer and the jPod intarsys PDF Renderer

**Swinglabs PDF Renderer**

Like the name already suggests this project is made to view PDF documents. It does not support any manipulation of a PDF document or anything else.

Hence PDF Renderer is only made to view PDF documents there is no overhead of unused libraries [25].

Swinglabs PDF Renderer offers a nice and easy way to render a PDF document into a Swing application. This is done by providing a modified JPanel where the loaded PDF document is rendered to [26].

**jPod intarsys PDF Renderer**

jPod is also pure PDF renderer [27]. The jPod PDF Renderer is based on the jPod PDF Library which is a framework for interaction with PDF files. Hence the jPod PDF Renderer and the jPod PDF Library are using the same classes for PDF files it is possible to manipulate a PDF file with the PDF Library and then display it with the jPod PDF Renderer. But the PDF Renderer does not require the PDF Library.

To embed a PDF file into a GUI jPod provides a renderer class which needs a Java graphic object. In order to provide such a graphic object the paint method of a GUI component has to be overridden.

The biggest problem with jPod is the license. Hence it’s under GPL use within a commercial application is not possible.
Discussion

Although both solutions are working quite well the Swinglabs PDF Renderer has a clear advantage. The 
provided pre initialized Swing components make the usage easier then the usage of jPod.

Aside from that the usage of jPod is prevented by its GPL license.

Based on that the Swinglabs PDF Renderer is the right choice for this application
Office Documents
To display office documents in Java there are basically 3 solutions available which support a wider range of formats, the Apache POI project\textsuperscript{16}, the OpenOffice API\textsuperscript{17} and the ODF Toolkit\textsuperscript{18}.

OpenOffice API
The idea behind the OpenOffice API is to offer the abilities of OpenOffice to the developer. OpenOffice is therefore used as a service provider and has to be installed on the system \cite{28}. While using the OpenOffice API an instance of OpenOffice is started and runs in the background.

Hence OpenOffice has to be installed additional 400-450 MB of disk space are required \cite{29}.

The OpenOffice API Java part offers a JavaBean which can be easily used to embed an OpenOffice window into a Swing application.

OpenOffice supports all of the requested office formats.

Apache POI
The goal of the Apache POI project is to offer a Java API to read and write Microsoft OLE (Object Linking and Embedding) based documents, which means Microsoft Office Documents up to Office 2007 \cite{30}.

It is announced that in future versions the new Microsoft Office Open XML format which is used since Office 2007 will be supported \cite{30}.

On the file reading side POI works as a parser for the Office files. It offers no renderer to paint them into a Swing GUI \cite{31}. Therefore an additional renderer has to be developed.

While POI works quite well with Excel files it still has problems with Word files. In addition to that a lack of developers caused the development of the Word part to be stopped for the moment \cite{32}.

ODF Toolkit
The ODF Toolkit is created to read and write all Open Document Format (ODF) files and give an easy access to them using a DOM structure \cite{33}.

An additional installation of OpenOffice is not needed.

Hence the ODF Toolkit only provides a parser for ODF files an additional renderer has to be developed.

Discussion
Since the ODF Toolkit and the POI project require an additional renderer that has to be developed additional time would be required for this.

\textsuperscript{16} For more information visit \url{http://poi.apache.org}
\textsuperscript{17} For more information visit \url{http://api.openoffice.org}
\textsuperscript{18} For more information visit \url{http://odftoolkit.openoffice.org}
The biggest problem with the OpenOffice API is the requirement to have OpenOffice installed and running which requires around 450 MB of additional disk space and around 160 MB of memory usage.

But with the offered JavaBean and the support for all the required office formats it’s the best solution as of today.

In future the development of an office file renderer based on POI and the ODF Toolkit should be considered and the OpenOffice API should be replaced by that.
Layers
The design of the Output Controller was based on a multilayer system allowing small directions hints or other small markers to be displayed on top of the actual content.

Java has some build in functions to realize multiple layers in a GUI. All these functions are based on the concept of what gets painted first.

This works well with static media like pictures or text but it doesn’t work with videos. Hence a video is painted ca. 25 times per second it simply just paints over any overlaying layer regardless the previous order.

The main problem here is that painting components works only in one way where the parent component tells all children that they have to paint themselves. There is no communication between the children or from a child to the parent [8]. This means that no component of the GUI would recognize if it is over painted.

To avoid this over painting there two possibilities

1. Synchronizing the paint methods
2. Let the operating system take care of

Hence synchronizing the paint method would require a complex redesign of a GUI component and it’s most likely that flicker effects appear thus that idea was dropped.

The possibility to let the operating system take care of this problem is available since Java 6 Update 10 where Sun introduced the AWTUtilities [34] which allow transparency for the top level GUI components, like JWindow or JFrame. Hence these top level components are controlled by the operating system the transparency has to be supported by the operating system.

Hence top level components are rendered and controlled by the operating system they and their children are not getting over painted by an underlying component [15].

Based on that and the new feature to have transparency for this top level components they can be used as a protection layer which separates the actual layers and their painting methods.
**Scenarios**

**Add new content to screen**

In order to add new content to an Output Controller Client the CMF sends a notification to the SelfSubscription object. The submitted data in this notification gets extracted and is send to the ScreenManager to add the new content to the screen.

Next the ScreenManager creates a new MediaContent object according to the submitted MIME type, in this example a VideoContent object.

Subsequent this MediaContent object creates the actual visual component that is being displayed.

Finally the ScreenManager adds the new content to its display queue and starts to manage the content on queue and the content on screen.

![Sequence diagram](image_url)

*Figure 24 sequence diagram: add new content to screen*
Manage Screen

In order to manage the MediaContent objects which are on queue or on screen the ScreenManager determinates according to their priority whether a MediaContent object on screen needs to be removed or a MediaContent object on the queue needs to be displayed on the screen. An object that needs to be displayed on the screen provides its visual component to the ScreenManager. Subsequent the ScreenManager fits in the visual component into the screen. Therefore the ScreenManager first detects the free space on the screen and subsequently tries to get the best fit. If necessary the visual component is resized before adding it to the screen.

Figure 25 sequence diagram: manage screen
**Testing and Profiling**

**Profiling Environment**
- CPU: AMD Athlon X2 4800+
- RAM: 2 GB
- Operating System: Windows XP Service Pack 3
- Java: Java Development Kit 1.6.0.11
- Internet connection: 100 MBit
- Testing and Profiling Software: Java Runtime Analysis Toolkit 1.0 beta 1

- Used test content:
  - Content1: http://samples.mplayerhq.hu/Divx4-bugs/Fight.Club.Trailer.avi
  - Content2: http://i.realone.com/assets/rn/cms/2005/games_tiny_square/dinerdash_50x50jpg.6957763.jpg
  - Content3: http://www.japaneseculturecenter.com/images/image-kempo.png

**Average Profiling Results Pixel By Pixel Method**

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th># of method calls</th>
<th>Average ms</th>
<th>Total Method ms</th>
<th>Method Time %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>main</td>
<td>1</td>
<td>7,425.00</td>
<td>453</td>
<td>6.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>addContent</td>
<td>3</td>
<td>2,324.00</td>
<td>16</td>
<td>.2</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>manageScreen</td>
<td>3</td>
<td>2,006.33</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>getFreeSpace</td>
<td>6</td>
<td>990.17</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>pixelUsed</td>
<td>6,535,602</td>
<td>0.0</td>
<td>4,349</td>
<td>57.5</td>
</tr>
<tr>
<td>Rest</td>
<td>Rest</td>
<td>-</td>
<td>-</td>
<td>2,607</td>
<td>-</td>
</tr>
<tr>
<td>Overall</td>
<td>Overall</td>
<td>-</td>
<td>7,425.00</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

19 For more information visit http://jrat.sourceforge.net
## Average Profiling Results New Method

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th># of method calls</th>
<th>Average ms</th>
<th>Total Method ms</th>
<th>Method Time %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>main</td>
<td>1</td>
<td>1,124.00</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>addContent</td>
<td>3</td>
<td>374.67</td>
<td>15</td>
<td>1.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>manageScreen</td>
<td>3</td>
<td>62.33</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>getFreeSpace</td>
<td>6</td>
<td>0.00</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>getCornerPoints</td>
<td>6</td>
<td>0.00</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>getVertMaxRectangle</td>
<td>12</td>
<td>0.00</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>ScreenManager</td>
<td>getHorizMaxRectangle</td>
<td>12</td>
<td>0.00</td>
<td>0</td>
<td>.0</td>
</tr>
<tr>
<td>Rest</td>
<td>Rest</td>
<td>-</td>
<td>-</td>
<td>1,109.00</td>
<td>-</td>
</tr>
<tr>
<td>Overall</td>
<td>Overall</td>
<td>-</td>
<td>-</td>
<td>1,124.00</td>
<td>-</td>
</tr>
</tbody>
</table>
5. Demonstration

5.1. Demonstration Environment
The environment for the demonstration consists of

- A server running the Context Management Framework, the Output Controller and providing media
- A RFID reader connected to the Context Management Framework
- A Bluetooth dongle connected to the Context Management Framework
- An eeePC with Linux running the Output Controller Client
- A NEC FullHD display connected to a Laptop with Ubuntu 9.04 running the Output Controller Client
- A HTC Diamond Smartphone running Windows Mobile 6.1 with Bluetooth

5.2. Demonstration Scenario
In the scenario for this demonstration are 3 different levels of personalized content demonstrated. Therefore 3 policies and are defined. Aside from that a commercial runs in an endless loop.

Figure 26 Screenshot Output Controller Client Demo. Default commercial.

The first policy is used for the content with lowest level of personalization. Here the show content will be based on the location of the display and the current time. So for instance a display located at the entrance at 12 o’clock would show the available restaurants in the area.
For the second policy the Bluetooth dongle was used to detect mobile devices in the area near the display it was connected. If a device reaches the area the available Bluetooth services will be displayed on the display on the second layer.

The highest level of personalization in this demonstration is reached using the RFID reader. Based on the read out ID the person in front of the display is exactly identified. With this information content specified for this person will be shown. In this demonstration the personal calendar is shown.
6. Conclusion and outlook

6.1. Conclusion

In this thesis, the design and development of a system that provides a threefold advantage is presented. Firstly, it includes a policy controlled content management system which can be used generically in various use cases. Secondly, it provides a content displaying system which can display the most common types of media and manage several contents on one screen. Finally, the development of a communication interface that uses a set of metadata to describe content and how it should be displayed is described.

Throughout the thesis, it is shown the applicability of such a system in multiple scenarios, making a clear case for the Digital Signage area. With this in mind, the demonstration section shows how this system can be directly used for such tasks.

The work described in this thesis provides information about how a policy based system can be designed and implemented and how the communication between a server and client can be established using metadata. Aside from that the thesis shows the complex problem and a solution of managing various contents on one screen.

6.2. Outlook

In today’s world mobile communication is already part of everybody’s life and mobile devices are already powerful enough to display videos, pictures and almost every common media type. Therefore it should be considered to develop an Output Controller Client for these mobile platforms.

Hence the Context Management Framework provides a communication interface via XML Remote Procedure Call (XML-RPC) various versions of the Output Controller Client can be developed technology independent.

Almost every mobile devices today even gaming handhelds have some kind of signal transmitter like Wi-Fi or Bluetooth which can be used by the Output Controller as context information. This combined with a running Output Controller Client on such a device makes the system capable of providing location based services.

In an actual product privacy will maybe a problem. This has to be considered in future developments. Privacy laws have to be fulfilled. Aside from this the user acceptance of such a service which could be abused to track people and construct illegal profiles might be a problem and should be taken serious in future developments.

Aside from extending the usability for digital signage this system could be used in various working areas. With some modifications on the metadata and a specialized version of the Output Controller Client this system can be used to control certain devices. This could be used from assisted living over the security business to even military use.
For instance the system could have various sensors connected controlling automated phone calls. This could be used in assisted living for quick responses on medical emergencies or in the security business for quick responses on security alarms.

Although there was no use case known that required a logical NOT operator for the combination of several subscriptions, it should be considered to implement such an operator. This would provide a complete Boolean logic.
6.2.1. Technical Recommendations
Within the Output Controller Client the usage of some libraries should be reconsidered.

Hence the OpenOffice API is designed to provide a very wide range of functionalities for interaction with OpenOffice and only one of these functionalities is really used results in a bit oversized solution for displaying office documents. In addition to that the requirement to have an OpenOffice installed and running while using the API requires around 450 MB of disk space and around 160 MB of RAM which leads to a bad cost – performance ratio. Considering these facts the OpenOffice API should be replaced by a more specialized library with a better cost – performance ratio.

Even through the Java Media Framework with the FOBS plug-in is the best solution as of today to display a wide range of videos does not make it the best solution forever. On one side FOBS itself has some bugs on the other side there are other upcoming technologies like JavaFX or the announced new Java Media Framework which show quiet promising approaches to give Java video functionality.
7. Bibliography


8. Appendix

8.1. Supported file format list for Java Media Framework

<table>
<thead>
<tr>
<th>Media Type</th>
<th>JMF 2.1.1 Cross Platform Version</th>
<th>JMF 2.1.1 Solaris/Linux Performance Pack</th>
<th>JMF 2.1.1 Windows Performance Pack</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIFF (.aiff)</td>
<td>read/write</td>
<td>read/write</td>
<td>read/write</td>
</tr>
<tr>
<td>AVI (.avi)</td>
<td>read/write</td>
<td>read/write</td>
<td>read/write</td>
</tr>
<tr>
<td>GSM (.gsm)</td>
<td>read/write</td>
<td>read/write</td>
<td>read/write</td>
</tr>
<tr>
<td>HotMedia (.mvr)</td>
<td>read only</td>
<td>read only</td>
<td>read only</td>
</tr>
<tr>
<td>MIDI (.mid)</td>
<td>read only</td>
<td>read only</td>
<td>read only</td>
</tr>
<tr>
<td>MPEG-1 Video (.mpg)</td>
<td>-</td>
<td>read only</td>
<td>read only</td>
</tr>
<tr>
<td>MPEG-1 Video (.mpg)</td>
<td>read only</td>
<td>read/write</td>
<td>read/write</td>
</tr>
<tr>
<td>Source: <a href="http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/formats.html">http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/formats.html</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.2. Supported file format list for FFmpeg

<table>
<thead>
<tr>
<th>Name</th>
<th>Encoding</th>
<th>Decoding</th>
<th>Name</th>
<th>Encoding</th>
<th>Decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>4X Movie</td>
<td>X</td>
<td>X</td>
<td>MPEG-1/2 video XvMC (X-Video Motion Compensation)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>8SVX exponential</td>
<td>X</td>
<td>X</td>
<td>MPEG-1/2 video (VDPAU acceleration)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>8SVX Fibonacci</td>
<td>X</td>
<td>X</td>
<td>MPEG-2 video</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>American Laser Games MM</td>
<td>X</td>
<td>X</td>
<td>MPEG-4 part 2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AMV Video</td>
<td>X</td>
<td>X</td>
<td>MPEG-4 part 2 Microsoft variant version 1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Apple MJPEG-B</td>
<td>X</td>
<td>X</td>
<td>MPEG-4 part 2 Microsoft variant version 2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Apple QuickDraw</td>
<td>X</td>
<td>X</td>
<td>MPEG-4 part 2 Microsoft variant version 3</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Asus v1</td>
<td>X</td>
<td>X</td>
<td>Nintendo GameCube THP video</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Asus v2</td>
<td>X</td>
<td>X</td>
<td>NuppelVideo/RTjpeg</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ATI VCR1</td>
<td>X</td>
<td>X</td>
<td>On2 VP3</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ATI VCR2</td>
<td>X</td>
<td>X</td>
<td>On2 VP5</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Autodesk Animator Flic video</td>
<td>X</td>
<td>X</td>
<td>On2 VP6</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Autodesk RLE</td>
<td>X</td>
<td>X</td>
<td>planar RGB</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AVS (Audio Video Standard) video</td>
<td>X</td>
<td>X</td>
<td>Q-team QPEG</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Beam Software VB</td>
<td>X</td>
<td>QuickTime 8BPS video</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bethesda VID video</td>
<td>X</td>
<td>QuickTime Animation (RLE) video</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Brute Force &amp; Ignorance</td>
<td>X</td>
<td>QuickTime Graphics (SMC)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C93 video</td>
<td>X</td>
<td>QuickTime video (RPZA)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CamStudio</td>
<td>X</td>
<td>Raw Video</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Chinese AVS video</td>
<td>X</td>
<td>Real Video 1.0</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Delphine Software International CIN video</td>
<td>X</td>
<td>Real Video 2.0</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Cinepak</td>
<td>X</td>
<td>Real Video 3.0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cirrus Logic AccuPak</td>
<td>X</td>
<td>Real Video 4.0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creative YUV (CYUV)</td>
<td>X</td>
<td>Renderware TXD (TeXture Dictionary)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dirac</td>
<td>E</td>
<td>E</td>
<td>RL2 video</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DNxHD</td>
<td>X</td>
<td>X</td>
<td>Sierra VMD video</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Duck TrueMotion 1.0</td>
<td>X</td>
<td>Smacker video</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duck TrueMotion 2.0</td>
<td>X</td>
<td>SMPTE VC-1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DV (Digital Video)</td>
<td>X</td>
<td>X</td>
<td>Snow</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Feeble Files/ScummVM DXA</td>
<td>X</td>
<td>Sony PlayStation MDEC (Motion DECoder)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic Arts CMV video</td>
<td>X</td>
<td>Sorenson Vector Quantizer 1</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Electronic Arts TGQ video</td>
<td>X</td>
<td>Sorenson Vector Quantizer 3</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic Arts TQI video</td>
<td>X</td>
<td>Sunplus JPEG (SP5X)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic Arts TGI video</td>
<td>X</td>
<td>TechSmith Screen Capture Codec</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Escape 124</td>
<td>X</td>
<td>Theora</td>
<td>E</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FFmpeg codec #1</td>
<td>X</td>
<td>X</td>
<td>Tiertex Limited SEQ video</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Flash Screen Video v1</td>
<td>X</td>
<td>X</td>
<td>VMWare Screen Codec / VMWare Video</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Flash Video (FLV)</td>
<td>X</td>
<td>X</td>
<td>Westwood Studios VQA (Vector Quantized Animation) video</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Fraps</td>
<td>X</td>
<td>X</td>
<td>Windows Media Video 7</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H.261</td>
<td>X</td>
<td>X</td>
<td>Windows Media Video 8</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H.263 / H.263-1996</td>
<td>X</td>
<td>X</td>
<td>Windows Media Video 9</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>H.263+ / H.263-1998 / H.263 version 2</td>
<td>X</td>
<td>X</td>
<td>Wing Commander III / Xan</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10</td>
<td>E</td>
<td>X</td>
<td>Winnov WNV1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10 (VPDAU acceleration)</td>
<td>E</td>
<td>X</td>
<td>WMV7</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HuffYUV</td>
<td>X</td>
<td>X</td>
<td>YAMAHA SMAF</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HuffYUV FFmpeg variant</td>
<td>X</td>
<td>X</td>
<td>ZLIB</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IBM Ultimotion</td>
<td>X</td>
<td>X</td>
<td>Zip Motion Blocks Video</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>id Cinematic video</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codec/Encoder</td>
<td>id RoQ video</td>
<td>Intel H.263</td>
<td>Intel Indeo 2</td>
<td>Intel Indeo 3</td>
<td>Interplay C93</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-------------</td>
<td>-------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Source: <a href="http://ffmpeg.org">http://ffmpeg.org</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>