

Intelligent objects by using RFID
&
Web technology

Decaestecker Jasper and Lapere Arend
Katho Kortrijk

June 22, 2011

Contents

1	Voorwoord	6
2	Description project	7
2.1	Introduction	7
2.2	Smart objects	7
2.3	Jointspace	7
2.4	TouchaTag (RFID)	8
3	Televic	9
3.1	Introduction	9
3.2	Story of the statue	9
3.3	History	10
4	Architecture	11
5	RFID	12
5.1	What?	12
5.1.1	Advantages	12
5.1.2	Disadvantages	12
5.2	How?	12
5.3	RFID tags	13
5.4	RFID reader	13
5.5	Alternate RFID reader	13
5.6	PN532	14
6	Programming RFID	15
6.1	Choices,Choices...	15
6.1.1	TouchaTag Service	15
6.1.2	WinScard.dll	15
6.1.3	Java SmartCardIO API	15
6.1.4	Libnfc	15
6.2	Libnfc	15
7	Code RFID	16
7.1	C/C++ - Libnfc	16
7.1.1	Setup	16
7.1.2	Reading tags	16
7.1.3	Counting tags	16
7.1.4	Sending event	17
7.2	C#	17
7.2.1	setup	17
7.2.2	Setting up the reader	17
7.2.3	Setting up the ConnectX client	18
7.2.4	Tag added event	18
7.3	Sleep mode	20

8	Visualization	21
8.1	Introduction	21
8.2	Different Approaches	21
8.2.1	dlna or not dlna that is the question	21
8.2.2	JointSpace	21
8.3	Alternatives	22
9	Implementation	23
9.1	Introduction	23
9.2	Basics	23
9.2.1	A first glance at VisualX	24
10	VisualX	25
10.1	XApplication	25
10.2	XWindow	25
10.3	XRectangle	26
10.4	XTextBox	26
10.5	XBrowser	26
10.5.1	Internet Explorer	27
10.5.2	Google Chrome	27
10.6	XButton	28
10.7	XSlideShow	29
10.8	XContainer	29
11	ConnectX	31
11.1	ConnectXServer	31
11.2	ConnectXDigitalSignageReader	32
12	VisualX insight	35
12.1	DirectFB	35
12.2	Drawing	38
12.3	A walk through GDI	39
12.4	A walk through DirectX	39
12.5	Improving VisualX	39
12.6	Colors and alpha-channels	42
12.7	Conclusion	42
13	Televic Signage	43
13.1	Digital signage in general	43
13.2	Televic Signage	43
13.3	Setup Televic Signage	43
13.4	Adjustments Televic Signage	43
13.4.1	Changing visual experience	44
13.4.2	Tag management	45
13.4.3	Person management	45
13.4.4	Plugin management	46
13.4.5	Activities	47
13.4.6	RSS feeds	48
13.4.7	RSS feeds historical & technical	48
13.5	The Code behind	48
13.5.1	Showing content	48

13.5.2	Deleting content	49
13.5.3	Zip	50
13.5.4	Appbrowser	50
14	Plugin System	51
14.0.5	TV guide	52
15	How everything works?	53
16	The Future	56
17	Field trips	56
17.1	Philips Brugge	56
17.2	IBBT Gent	56
18	Complications	56
18.1	Compiling libnfc	56
18.2	Televic Signage	57
18.3	The power of green computing	57
18.4	Direct FB	57
18.4.1	Drawing images with alpha channel made glitches	57
18.4.2	Connection cannot be closed	57
18.4.3	Connection with tv broken	58
18.4.4	Enumerating televisions	58
18.4.5	Browser crash	58
19	Televic Demo day	59
20	Conclusion	60
21	Bibliography	61
21.1	Information links	61
21.2	Software	61

List of Figures

1	Statue "Loud Silence"	9
2	Architecture of project	11
3	VisualX Logo	25
4	Internet Explorer logo	27
5	Chrome Logo	27
6	VisualX Logo	31
7	Stride in memory	38
8	Difference in sign of stride	38
9	view of people management page	46
10	view of the plugins page	47
11	view of calendar page	47
12	Main menu of the Tv Guide	52
13	Start of the programl	53
14	Main menu	53
15	Album	53
16	Picture browserl	54
17	Activity managerl	54
18	Feed browser	54
19	Browsing through the articlesl	55
20	Application browser	55
21	A plugin at work	55
22	There was much rejoice !!!	59

1 Voorwoord

Stepstone to a brighter Yesterday.

Onze opleiding te KATHO laat ons toe om ons eindwerk en stage te combineren. Dit is een prachtige opportuniteit, aangezien deze een nauwlettende ervaring weergeeft met het werkveld. Wij hebben gekozen voor dit project aangezien we beide genteresseerd waren in het ontdekken van de RFID technologie en het verder uitwerken van onze kennis i.v.m. het programmeren op het web.

Graag schenken we een dankwoord aan:

- prof. dr. ir. Piet Verhoeve en dr. ir. Brecht Stubbe, onze buiten-promotoren, die ons hebben geholpen bij al onze vragen en het helpen uit werken van ons project.
- Dhr. Philip Vanloofsvelt, die ons geholpen heeft bij onze vragen.
- Het bedrijf Televic NV, die het mogelijke maakte om dit prachtige project waar te maken.
- Onze ouders, die ons goed hebben verwend tijdens de harde werkdagen te Televic.
- Urshi, de vriendin van Arend, voor het maken van onze RFID¹ kubus.

Zoals op te merken is, is ons voorwoord in het Nederlands geschreven, dit was een gegeven vanuit het KATHO. Omdat Televic een multilinguaal bedrijf is, wordt ervoor gezorgd dat iedereen in staat is deze thesis te lezen, daarom wordt alle hieropvolgende tekst van de thesis in het Engels geschreven.

¹RFID = Radio-Frequency Identification.

2 Description project

2.1 Introduction

The main assignment of our project was to research the possibilities of the RFID technology in combination with web technology. Although RFID isn't a brand new technology, it's not used in the context similar to our project. If we combine an ordinary object, with a RFID tag, we have created a smart object. To have interaction with this smart object and the environment, a smart system must be developed. With these objects we could be able to control an application on an alternative way. To be able to interact with the user, a visual GUI² must be made, which is being displayed on a television.

The application that would be running on this television, is a multimedia platform, which is able to show content, according to the user which is controlling this system. The idea behind this platform was inspired by a research concept that they are running in the "Woonzorgcentrum De Vijvers". The concept was created by Televic and IBBT in the context of the TransCare project. This application, made by the IBBT in Gent, enables the users to see messages and pictures of their family and friends. The main advantage of this platform, is that the users can still use their television, which they are familiar with and access information, which would else be only accessible by using a computer.

2.2 Smart objects

There are 2 different categories of smart objects. A smart object can literally be a "smart object", that is capable to interact with the environment using his own computing power, to determine the outcome of events that take place in the world around it. This type of smart object can be compared to a person, which has the capabilities to make up his own mind.

The second type of smart object is far from smart, the environment itself is smart. The smart object itself can do nothing, the only purpose is to identify the object to the system. The "smart" object has only been given an id, the smart environment around it will detect this id, as this object moves through the plane. According to the id of the object, the system will trigger different events. The events are stored in a database, which are linked to the id of the tag.

For our project we use the latter category of smart objects.

2.3 Joinspace

Of course abstract data isn't of much use when you can't really do something with it. Every system's downfall is the design of bad interface. Your software may be so brilliant, if one can't wield it's power through an easy to use interface, no one will use it. Our goal was to use a television to represent our data. That's where joinspace enters the picture, a technology developed by Phillips Brugge to show custom data on a television.

²GUI= Graphical User interface

2.4 TouchaTag (RFID)

We can control our application in a traditional way, by using a standard remote control. Although most of the times, these controllers contain way to many buttons to be used with ease. By using RFID technology we could be able to do the basic "computer" tasks with the use of everyday objects. Imagine that watching your photo album could be done by laying a picture on a reader. This is much easier than starting up a computer and trying to find the directory where you placed your pictures, not to mention the the need to connect your computer with a large TV screen, so that everyone can enjoy watching your album.

3 Televic

3.1 Introduction

Televic is a company who develops and produce high quality communication systems. Televic insures quality by deploying products that is made to the costumers' wishes. Televic works with the latest state-of-the-art technology which can only be achieved by innovation. The employees of Televic are passionate about their job and excel at their field of expertise. Televic is a financial independent group, who are are working hard to be respected around the globe. Currently Televic has over 400 employees, which are employed in Belgium, France, Great Britain, Bulgaria and China. Televic Headquarter is stationed in Izegem, in this site they mostly concentrate on development of new products. The first prototypes of a new product are made here and are thoroughly tested. This ensures that every product that is released onto the market, meets the strict requirements needed in the field of communication systems.

Televic has 5 branches were they deliver theses systems:

- Audiovisual
- Rail
- Health Care
- Conference
- Education

3.2 Story of the statue

The statue "Loud Silence" is the symbol for Televic.

Loud symbolizes the need for communication and the need to be heard by others.

Silence stands for efficient communication by using the appropriate solutions or technologies.

The contradiction between Loud & Silence is symbolic for a good balance between the need for communication and the need to organize this communication in an efficient way. Televic is specialized in providing solutions for critical & interactive communication. "Loud Silence" is a statue made by the sculptor Guy Timmerman.



Figure 1: Statue "Loud Silence"

3.3 History

Televic was founded in Belgium, Roeselare by Mr Van Hulle in the year 1946. Initially, Televic started out manufacturing radio receivers. The giants in consumer electronics, however, quickly colonized this market and the company decided to focus exclusively on professional systems.

In 1953 the first nurse call system was introduced, this system called "T.C.D" had integrated intercom capabilities. It wasn't until 1965 till the company moved to Izegem/Kachtem, which is still the headquarters of Televic.

During the golden sixties Televic started producing high quality audio product like loudspeakers, mixes and PA systems for use on stage. Televic has outgrown Belgium and were deploying these systems around Europe.

In the 70's they introduced nurse call systems, which can be compared with technology found in contemporary hospitals. The new bed unit made it possible for the patient to control music programmes and lights from the bedside. In the late seventies Mr. G. Maes becomes the new CEO/owner of the company.

With the launch of the T.N.C nurse call system, Televic was one of the first manufacturers worldwide, to have a processor-based, bus cabled and computer controlled nurse call system. In cooperation with the university of Leuven, a single-chip microprocessor was developed, the UNIF-05. this system was very successful during the eighties. The T.N.C. system is still used in present-day hospitals.

In the late eighties Televic acquires the company Baert P.V.B.A.. This company also develops, manufactures and markets training systems and language laboratories under the brand name ARTEC. The product lines from Televic and ARTEC become one. This is the start of a new range of multimedia learning systems.

During the nineties they deployed a digital controlled conference system in the new European Parliament, in Brussels. A first introduction of the network-based nurse call system called "AXIO". At the end of the 1st millennium, Mr Lieven Danneels and Mr Thomas Verstraeten take over Televic from Mr Maes as owners/CEOs.

A new millennium, new opportunities, Televic celebrates its 60th anniversary. Although Televic is already known around the globe, it wasn't until 2007 that they build manufacturing plants outside of Europe. In 2007 they opened a manufacturing plant in Bulgaria and in 2010 a plant is opened in china.

Televic has won an award from the Flemish Government for "most promising Company" in 2005.

4 Architecture

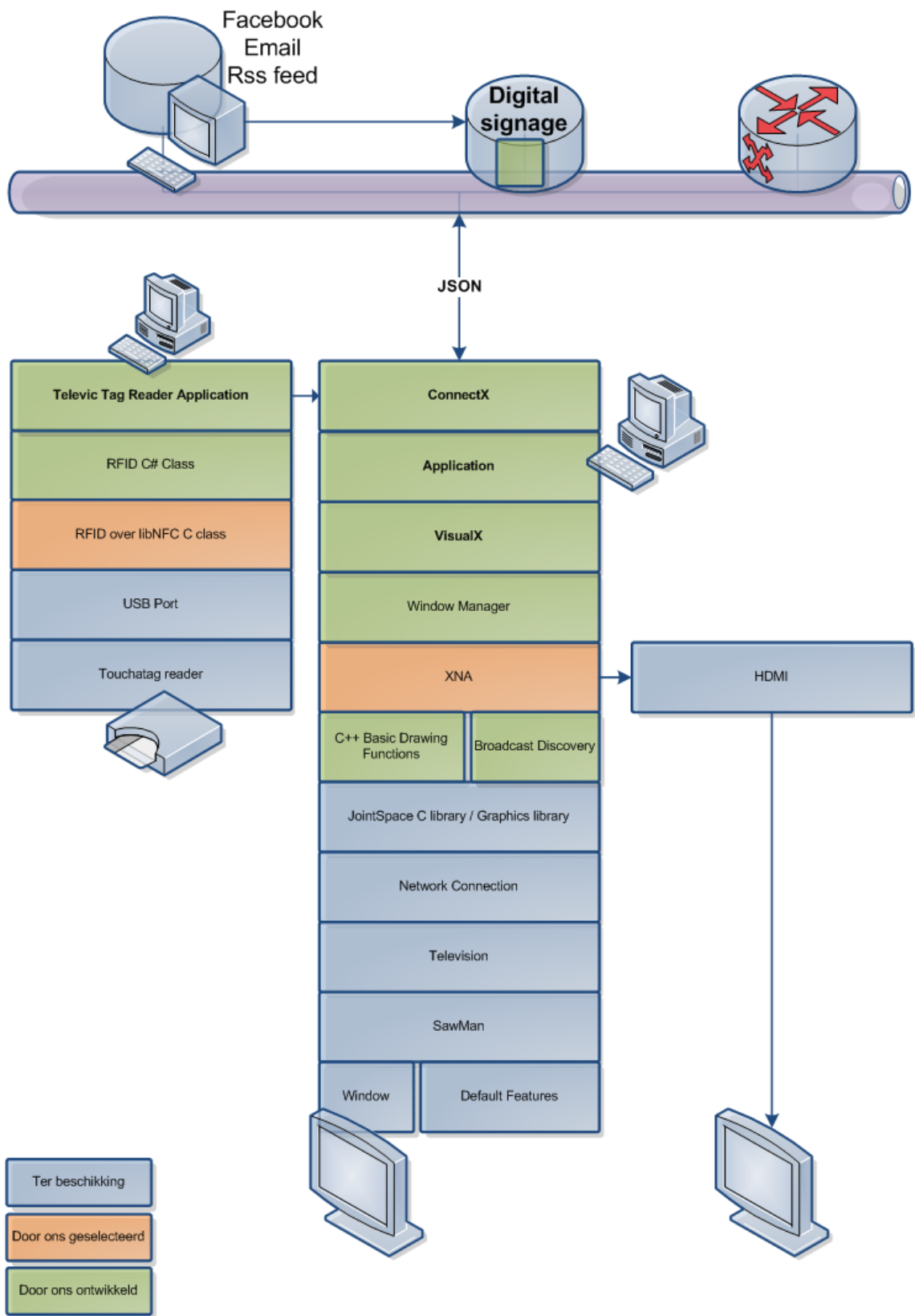


Figure 2: Architecture of project

5 RFID

5.1 What?

RFID = Radio-Frequency Identification. RFID is a technology to store and read information between a RFID reader and a RFID tag. Communication is done by using radio-frequency waves. At the moment RFID is used for identification and localisation of objects. Some examples of RFID usage in our daily lives are: Anti-theft detection in shop (mainly clothing), localisation of packets that are send with a delivery company, identification of cattle and registration of house pets. In our project we will use it for identification of objects.

5.1.1 Advantages

In the future it is possible if not certain that the RFID technology will make the bar code obsolete. RFID tags can be scanned from a further distance than bar codes. No visual contact is required to scan these tags. With bar codes, it is only possible to scan on code at a time. With a strong enough reader and the right software, multiple RFID tags can be read at a time.

5.1.2 Disadvantages

Since RFID uses radio-waves, they can be easily disrupted using energy at the right frequency. When the signal of multiple readers overlap, this could cause reader collision. The tag isn't able to respond to simultaneous queries. When multiple tags are present in a small area, RFID tag collision may occur. At the moment it is cheaper to print a bar code, than to make one RFID tag, although through mass production and innovations, it becomes cheaper to make. The biggest disadvantage may be privacy issues. In the future it may be possible that humans are implanted with a RFID chip, which could replace the need of a passport. Since RFID readers are cheap and available everyone could scan youre identity, without you even knowing.

5.2 How?

The system has 3 important parts:

- an antenna
- a RFID tag
- a transceiver

In most cases the transceiver and the antenna are implemented into one device, which is called the reader. The antenna will send out radio-frequency signals in a relative short range. The purpose of this signal is:

- Providing communications between the reader and the RFID tag
- Providing energy to the RFID tag

When a RFID tag passes through a signal that comes from the reader, it detects the activation signal. This signal will power up the RFID chip, and the tag will transmits its information. This information will be picked up by the scanning antenna. with the right software we can retrieve the id of the tag.

5.3 RFID tags

There are two types of RFID tags. You have active tags and passive tags.

The active tags have their own power source, the extra power will provide more functionality. The amount of features reflects the cost, which make active tags much more expensive.

In our case we use passive tags, these tags don't have batteries. These tags are very easy to produce and are very cheap. The frequency that is used for communication is 13.56 MHz. Note that due to the usage of radio frequencies, these tags do not function when they are used within close range of metal or liquids.

There different sorts of passive tags on the market. The most famous tags are MIFARE, these tags are made by NXP semiconductors. There are many different configurations of the these tags: MIFARE classic, MIFARE ultralight, MIFARE plus,... The tags from TouchaTag are MIFARE ultralight. These low-cost ICs employ the same protocol as MIFARE Classic, but without the security part and slightly different commands. They only have 512 bytes of memory. These chips are so inexpensive, they are often used for disposable tickets for events.

5.4 RFID reader

There are a variety of readers on the market. There are 2 major types of reader on the market:

- Embedded readers, mainly used in mobile phones.
- Separate readers, which in most times can be connected with USB/Ethernet.

For the embedded reader the most used MCU³ for NFC⁴ are produced by ARYGN technologies. There are multiple configurations of their MCUs, but only a few have been tested with the library we use for programming.

For separate readers, the ACR122 are the most used readers, these readers are produced by ACS. These reader also come in different products, with the main difference being alterations in the design. The TouchaTag reader that we use is a ACR122U102. All these readers are using a PN532 NFC controller chip and a ST7 microcontroller unit. All ACR122 reader are compatible with the Libnfc library, this enables us to use different readers, with the same driver to be used in our program, without the need to adjust the code.

5.5 Alternate RFID reader

Because we are using a RFID reader which is connected using USB, the distance between PC and RFID reader is limited to 5 meters. Therefore we can only implement the RFID technology if we provide a PC, close to the television.

A solution to this problem could be a RFID reader with Ethernet connection. Because of the time limit and the cost of these devices, we haven't tried this solution. This reader would not be compatible with the code we have written, so we would needed to invest even more time in developing the software. If in future development this implementation would be required, the WEB08S RFID reader could be used. this was the cheapest RFID reader with Ethernet connection available on the market. This reader acts as a HTTP client and can be programmed using PHP.

³MCU = Micro Controller unit

⁴NFC = Near field communication

Another option would be to use USB over Ethernet. There are different network-attached USB hubs available on the market, which enables us to connect USB devices, in our case the RFID reader on this hub. This hub would then communicate with a server using Ethernet. This solution would be a great solution, as this solution would work with our software. There is a main disadvantage however, as these USB to Ethernet hubs are very expensive. A 2 port model, costs 220 euros.

5.6 PN532

This is an integrated transmission module for contactless communication at 13.56 MHz, including micro-controller functionality. These are used for different kinds of passive contactless communication. This PN532 chip supports all MIFARE product, which include the Ultralight MIFARE tags that are used by TouchaTag. To make information exchange to the host systems, several interfaces are implemented:

- SPI
- IC
- Serial UART

The PN532 embeds a low dropout voltage regulator allowing the device to be connected directly to a battery as well as a medium power switch to supply and control the power of the companion secure chip. This chip is implemented inside the TouchaTag reader.

6 Programming RFID

6.1 Choices, Choices...

If we want to use our RFID reader, we will need some software that will communicate with the reader through the USB interface. We had different option we could choose from:

- **6.1.1 TouchaTag Service**

This service is given if you buy a TouchaTag reader. We quickly abandoned this path, because the service required a constant internet connection and required the use of their database. Programming could be done by using the TouchaTag API.

- **6.1.2 WinScard.dll**

This is the SmartCard API from windows. Due to lack of proper documentation, we only tried it out, but without any decent success.

- **6.1.3 Java SmartCardIO API**

API to communicate with SmartCard while programming in JAVA. This API was well documented and we succeeded in making a program that could read a tag, we also stopped walking this path, because we are not really familiar with Java programming.

- **6.1.4 Libnfc**

C/C++ API that allows communication to most NFC devices on the market. Including the TouchaTag reader. This API is being used by a lot of people, so documentation was really good. An advantage of this API is that the source code is written in C/C++. This enabled us to make a fast transition to C++.Net and finally C#. We chosen this path for programming the RFID communication.

6.2 Libnfc

Libnfc is a free library that can be used to communicate between the RFID reader and the application running on the computer. Libnfc can be build on all major OSs and is compatible with most of the NFC devices currently on the market. The most developers use Linux, this reflects that the library is better supported for Linux than windows. The library supports modulation for MIFARE, FeliCa, NFCIP and ISO14443 tags. This library is written in C/C++.

7 Code RFID

7.1 C/C++ - Libnfc

Before we show the Application code from C#, we will explain the most important parts of the C/C++ code. In the C/C++ code we make use of the Libnfc library, this code will take care of the communication between the application and the USB device. This code is the foundation of our tagreader application, and no changes should be made in this code.

7.1.1 Setup

Before we can start programming we need to add the Libnfc library to our project. When we compiled the source code from Libnfc, we generated several files. We need to add the folder "nfc" to our project. it is necessary to include the "nfc.h" file, which contains the prototype of the functions we will use.

7.1.2 Reading tags

With this code we are searching for tags. This code is run continually on a separate thread. If a tag has been found, we read the tag id and convert it to a hex format which has 7 pairs of 4bit hex value. This is the format that can be read on the TouchaTag tags. example: "048BB8F9232580".

```
if (nfc_initiator_select_tag(pnd,(nfc_modulation_t)0,nullptr,0,&nti)) //
    Search for tag
{
    strcs = genew String("");
    for(int i=0; i < 7;i++) // make string of hex value
    {
        dum[i] = new char[3];
        sprintf( dum[i], "%02X", nti.nai.abtUid[i] );
        strcs += genew String(dum[i]);
        delete dum[i];
    }
}
```

We want to be able to read multiple tags at the same time. each time a tag is read, we check if the current tag exist inside or tag list. If it doesn't exist, we add the new tag to the list.

After a certain time ± 200 ms we will send an event with the list of the current tags. if no tags are detected, no events are sent.

7.1.3 Counting tags

We want to know if we added or subtracted tags on the reader. Therefore we need to count the amount of tags on the reader. We also count the tags from the previous counting. The difference between the previous tags and the current tags will determine if tags are added or removed.

```
for(int i = 0 ; i < 10 ; i ++ ) // run through list of tags
{
    if (tags[i] != "")
    {
        cntTags++; // new tag detected
    }
    if (prevtags[i] != "")
    {
```



```

        cntPrevTags++; // old tag detected
    }
    diff = cntTags - cntPrevTags;
}

```

7.1.4 Sending event

Now we need to determine if it is required to send an event. We only need to send an event if the previous tags are different to the current tags. If they are the same, nothing has happened and no event must be send. If there is a change, we will send an event with the list of the tags that are on the reader. If the difference is a positive number then we have added a tag to the reader. If the difference is negative, then a tag has been removed. In our C# application we subscribe to these events and run the appropriate code according to the the information that the event has offered us.

```

for(int i = 0; i < 10 ; i++)
{
    if(tags[i] != prevtags[i]) // Check if previous tags are current tags
    {
        if (diff >= 1)
        {
            Raise(tags); // Tag added
        }
        else if(diff <= -1)
        {
            Lower(tags); // Tag removed
        }
        diff = 0; // reset difference counter
    }
}

```

After sending the events, we will store the current tags inside the previous tags. The current tag list is being cleared so it can be used for new readings. This C/C++ code is being added to our C# application by using a dll.

7.2 C#

7.2.1 setup

Before we can start to make our C# application, we need to add the CLRTagReader dll. This enables us to use the code we have written in C/C++ in our C# application.

7.2.2 Setting up the reader

In the code below we make a new tagreader, which is called TouchaTag. We immediately subscribe to the 2 events, the first event will be triggered if a tag is being placed on the reader, while the other event will detect tag removal. We also check if a Tagreader is connected. When a tagreader is not connected to a computer, a messagebox will notify the user.

```

private void SetupReader()
{
    touchatag = new tagReader(); // make new tagReader: touchatag
    int i = touchatag.checkConnection();
    touchatag.OnRaiseEvent += new
        RaiseEventHandler(touchatag_OnRaiseEvent); // subscribe to
        raise event = tag is placed on reader
}

```

```

touchatag.OnLowerEvent += new
    LowerEventHandler(touchatag_OnLowerEvent);    // subscribe to
    lower event = tag is removed from reader

if (i == 0)
{
    MessageBox.Show("Tag reader not connected");
}
}

```

7.2.3 Setting up the ConnectX client

Here we try to make a new ConnectXClient. The constructor of the object requires 3 parameters: name of the connection, classification of the connection and the Ip-address of the connectXServer. If we make a successful connection, the user will be notified of the success. If the server can't be found, no connection will be made, and the information read from the tags can't be send to the ConnectXServer. We will try and make connection with the ConnectX server each time a tag of a different user is detected. This is necessary because this enables us to show different data on the screen according to the person that is currently using the application.

```

private void SetupConnectX(string id)
{
    try
    {
        clientTv1 = new ConnectXClient(id, "Tv", "10.0.70.34");
        networkOnline = true;    // network connection is online
    }
    catch
    {
        networkOnline = false;    // network connection is offline
    }
}

```

7.2.4 Tag added event

If we want to add certain events to a tag, we need to store the information of the tags on a database. The data for each tag is stored in the Televic signage database. When a tag is detected, we check if the tag is stored in a local buffer. This buffer contains a list of all tagid's, linked to the user id and action. The buffer is filled each time a new tag is detected. If the tag exists inside the buffer, we retrieve the data from the buffer and send it to the connectX server. If the tag is not stored in the buffer, we will retrieve the data from the televic signage database instead. This new tag will then be stored in the buffer.

With the help of this buffer, we do not need to retrieve every tag from the database. This will improve speed of the system and will limit the data that is send over the network.

Before we can read or store information on the database, we need to connect and login to the Televic signage, with the login and password name of the company we want to login to. If connection is successful we should get a valid session. Hereafter we will get the information of the tag which is placed on the reader. We parse the data we have gotten from the Televic signage database and we determine the id of the person, which we call "id" and the id of the tag which is stored in index 1 of the tagId array. If the id of the person is different to the id of the the person of the previous tag, we will make

a new connection to the connectX server. When the id is the same, we will send the name of the event to the server.

```

private void TagAdded(string[] val) // Code is run when tag is added
to reader
{
    tagId = val[0]; // the selected tag is the first tag on the
    reader;

    if (Buffer.ContainsKey(tagId))
    {
        string id = Buffer[tagId].User;
        if (id != prevId || clientTv1.Connected == false)
        {
            SetupConnectX(id);
            prevId = id;
        }
        else
        {
            clientTv1.SendString(Buffer[tagId].Action);
        }
    }
    else
    {
        timer1.Enabled = false;
        timer1.Enabled = true;

        /* GET ACCES TO TELEVIC SIGNAGE */
        string server = "192.168.0.1"; // ip adres of signage
        string session_cookie;
        WebClient WC = new WebClient();
        WC.DownloadString("http://" + server +
            "/branch/manager/login.php");
        session_cookie = WC.ResponseHeaders["Set-Cookie"];

        WC = new WebClient();

        System.Collections.Specialized.NameValueCollection postData = new
            System.Collections.Specialized.NameValueCollection();
        postData.Add("username", "Admin");
        postData.Add("password", "password");
        postData.Add("company_name", "firefly");
        postData.Add("company_password", "mini");
        postData.Add("remember", "");
        postData.Add("submit", "");

        WC.Headers.Add("Cookie", session_cookie);
        WC.UploadValues("http://" + server + "/branch/manager/login.php",
            "POST", postData);

        session_cookie = WC.ResponseHeaders["Set-Cookie"];

        /** GET STUFF ***/
        WC = new WebClient();

        WC.Headers.Add("Cookie", session_cookie);
        string data = WC.DownloadString("http://" + server +
            "/branch/manager/tags.php?tag=" + tagId);
        if (data.Contains(", "))
        {
    
```

```

string cut = data.Substring(data.IndexOf("["),
    data.IndexOf("]") + 1);
cut = cut.Replace("\", "");
cut = cut.Replace("]", "");
cut = cut.Replace("[", "");

string[] event = cut.Split(new char[] { ',', ' ' });
string id = event[3];
if (id != prevId || clientTv1.Connected == false)
{
    SetupConnectX(id);
    prevId = id;
}
else
{
    clientTv1.SendString(event[1]);
}

Buffer.Add(tagId, new Tag() { Action = event[1], User = id });
}
}
}

```

Note that the information will not always be send to the connectX server when a tag is added. When a user is adding a cube to a person, the PHP code will not return any information from the database, but will instead insert the detected tagId into the database. The tagId, the current event and the id of the person will be added into the database. When a tag has been read that isn't detected in the database, nothing will happen.

7.3 Sleep mode

When a user is not using the TV application for a period of time, this user will be logged off. The user will be informed when the system is in "sleep-mode" and needs identification. This can be done by placing an object, with an RFID tag of course on the reader, the id of the person will be read and the system will be initialized with the settings for this persons. We have created this "sleep mode" by running a thread with a timer. When this timer has counted to the selected value, the connectX client will disconnect from the ConnectX server, and the Tv application will go in sleep mode. We also reset the id of the person, then we are sure that the next tag on the reader will make a new connection to the connectX server. Note that the Tagreader application never stops reading, this will keep running, else it would be impossible to detect a tag.

```

private void timer1_Tick(object sender, EventArgs e)
{
    timer1.Enabled = false;

    clientTv1.Disconnect();
    prevId = "";
}

```

8 Visualization

8.1 Introduction

Today we can choose between two technologies i.e. dlna and Philips' JointSpace. The latter one is a very young, but groundbreaking technology for just rendering stuff on a screen, whilst dlna reads the data from a serving host and then renders stuff on screen through it's software. In the next chapter will look at the differences.

8.2 Different Approaches

8.2.1 dlna or not dlna that is the question

dlna⁵, a widely adopted standard created by Sony. It provides a way for consumer products to communicate with each other and so creating a big media LAN. Say for example you have pictures and movies on a dlna certified NAS⁶. With a dlna certified television you can find that NAS, browse it and view its content on the television. dlna is a fast-expanding standard, and many company's create such devices. There's also a way to implement this on other non-dlna devices through software, so expect to see more of these devices in the future! No this is indeed a very powerful, interesting technology, but is it usefull for other things, like say creating an application? Well actually no. First of all we couldn't find much information about implementing dlna in your home-brew application, second of all nowhere is stated that you can actually read command's from a remote, our the other way around, send command's to the device, other than querying for media. So that's where we stopped the research on dlna and continued with JointSpace.

8.2.2 JointSpace

JointSpace is being developed by a very small group of engineers at the Philips company at Brugge (Belgium). It it build upon SPACE⁷. SPACE really splits the different parts of TV-software development. Where in the past engineers used to write everything themselves, they now offer a change to 3rd parties to create software for their televisions, which is actually a good thing! So they created JointSpace which is regular SPACE combined with DirectFB, a lightweight rendering API, which already had support for remote rendering through ethernet! In one of their earliest attempts to make JointSpace viable, they allowed to store a program (compiled with their compiler of course) on a USB stick. This program would auto-boot, but the user had no control over what it would do, so technically you could create a virus or other annoying program that would interfere with the normal behavior the end-user expects. So they locked that idea away, and a short time later they had another, better idea. Since almost all their televisions have an ethernet port, they could access the television through a network. They expose only limited resources to the television, so you can't just do whatever you want with it, so that partly increases security.

So they made it possible for us to create an application remotely, and we can control it with our television's remote control. This is a whole different approach from the one dlna has. Here the television can't really buffer, because it doesn't know what will come

⁵Digital Living Network Alliance

⁶Network Attached Storage

⁷Split Architecture

next, it's not a video stream or some sort, just plain old bitmaps that we send to the screen.

The big advantage of this system is that you can show whatever you want, because the television doesn't render anything! The big disadvantage is that you can show whatever you want, because the television doesn't render anything! Say there's a building filled with JointSpace enabled televisions, they're all activated, and they're all on the same network, well through JointSpace you can scan the network, and send a shutdown command to all the televisions! This is not what one would want to happen, so this is something that hopefully will change in the future. Like we said before, we have to render the images and then push it to the screen. This can seriously hog a network. Right now they use RLE⁸, a way to compress raw bitmap data on the fly. Personally I think they used this technique for proof of concept because it's easy to implement and it gives us an idea of what it does. They told us that in the near future they will use a far better compression technique, however this could drastically increase the CPU usage, while decrease network hogging. However it's still the best system for us. It changes our way of thinking where we used to need a computer next to each television to do these kind of things, now you technically need only one computer to serve multiple televisions. The main difference between dlina and JointSpace is that dlina pulls the data from a remote system, while JointSpace remotely pushed data to the screen.

8.3 Alternatives

Next to JointSpace we were also looking at alternatives, because you can't just force people to buy Philips TV's because of JointSpace. The alternative is the most hybrid way known to man, just attach a computer to the television. OK this might look easy, because we all know how to do this, however, we don't have the cool extras like JointSpace has, like overlays, shutting down the television or just return to the last visible TV-channel. One could omit this, by installing a decoder in the computer, and let the visual data from the TV-channel pass through while we draw a rectangle on top of it, with our application in it.

⁸Run-time Length Encoding

9 Implementation

9.1 Introduction

So we made two choices, JointSpace and the alternative way. We focused on JointSpace, since the alternative would be much easier to implement. In JointSpace, there is one important fact: it has limited graphical resources. Yes you can store an image on the television, and yes you can move that image around quite fluently. But it has only four megabytes of memory to store stuff in, and this just doesn't cut it. Certainly because we want to use high(er) resolution images. When the memory is full, the television just closes your application and that's it. So we had to figure out a way that we could show high resolution images, while preventing the television from closing our application. The second problem occurred while we were creating some test-applications. The test-applications on the JointSpace website can be compiled with the use of cygwin (a cross-compiler to compile Linux programs for windows), these programs work perfectly, and show the capabilities of JointSpace. However, these binaries don't play well with windows, and since cross-platform programming is not our field of expertise and the fact that there was a win32 port, we didn't continue with cygwin. The main problem with the win32 port is that a lot of the functionality of the full API isn't there. We can't print text on the television screen, we can't shutdown the connection and we can't enumerate the televisions in a network. These are only a few of the limitations we found rather annoying.

9.2 Basics

So we made our conclusion, do it yourself, or don't do anything at all. So we started creating a graphics library that would omit these problems. The main idea was to create a list of sprites, do stuff with them, render them to a bitmap, and send the bitmap to the screen. The first tests we did with this approach were as perfect as it was simple. However there was still the problem of the enumeration of the televisions. When I looked at the source code I could figure out the data-gram that was being sent for the broadcast, and on what port it was being sent. However when I broadcast this message, no one responded. So there was something wrong, the perfect solution to this problem was...capturing the broadcast packet with WireShark, from a program that was compiled with cygwin. We copied its data into a string and we send this over the network, and surprise surprise it did work!

OK, we solved the tiny network issue. But there is something fishy about the drawing functions don't you think? Exactly! Why render the whole image (1024x786x16 bits), and send the whole image to a television connected to a limited network, with limited resources? That's bit-suicide! Yes we know, the first-optimization we did was this, we kept a list with all the sprites, and then checked for the ones that were changed. If none changed, nothing needed to be drawn, and nothing had to be sent to the television. If one or more sprites changed, that one would be redrawn, and all the other sprites intersecting with that particular sprite. Only that part of the buffer-bitmap would be redrawn, sent over the network and redrawn on screen. A very huge resource saver. We implemented a simple form of dirty rectangles. It isn't a full implementation of dirty rectangles, because in a full implementation we would actually define the dirty rectangle, instead we redraw the whole image, even if only one pixel would have been changed.

9.2.1 A first glance at VisualX

At the time we didn't know where our approach would lead us, and what complications there were to come. VisualX is the name we gave to our drawing API. It contains the actual rendering to a (television)screen. But we gave it a lot more functionality. We found it to be rather hard to build an actual application with just a basic graphical-engine. It's like writing a game from the start: you'd have to do everything yourself. That gives you a lot of power, but it takes too long too actually get a working, bug-less system. We wrote a window-manager on top of VisualX. This window manager takes care of all the windows inside an application, which one should be drawn, which one should be visible, active and so on. The window-control is the part of the screen on which you can begin to draw controls. We've also made a few controls ourselves. These controls have been tested under every possible circumstance, ensuring that everything does what it should do. The currently available controls are: a button, a rectangle/an image, a container, a container-page, a window, a browser, a timer and a slide-show control. With these controls we can build almost every possible application. These controls are all derived from an abstract control class. Here we find all the properties and function that are the same for every control.

10 VisualX



Figure 3: VisualX Logo

10.1 XApplication

The XApplication class is our main class to which different windows will be hooked. Here we maintain the connection with our render-device (screen or television). Here is where we define the position and size of the screen to be drawn on television, for a normal screen it's defaulted to full-screen.

```
XApplication XA = new XApplication(this.Handle);  
//We pass the handle to the current program, stating  
//that the rendering will be done on the program's surface  
  
XA.ConnectToTV("xxx.xxx.xxx.xxx");  
//The ip-address of the television
```

10.2 XWindow

An XWindow is a window like the ones you know from your favorite OS⁹. The XWindow class already has a default control, the XRectangle control. This rectangle is the background of our window and can be user defined through a bitmap or a color. This class should be directly added to an application. A window is like a drawing board, this is the place where all the actual drawing will be happening. You can hide it, or show it. A window is never actually destroyed, except when explicitly asked. By default every XWindow (and all other controls for that matter) has the ability to be animated. The animation of a window is 'zooming' where the user-defined background zooms in (showing) or out (hiding). The length (or quality) of the animation is a simple variable. This variable just states how many steps there are to be taken between the previous state and the new state. Animations weren't just added to give the application a nicer touch, they're a extra feedback for the end-user, so he knows what's going on (or that is the big idea behind our animations).

I'd like to add here that only visible and activated windows can be used for interaction. You can't just go and work on another window, with the previous window still on top. This ensures that no multitasking can be done, which is a big difference to what we are used of. However the goal was not to develop a multitasking environment.

```
XWindow Window0 = new XWindow(new Rectangle( 0,0, 1024, 768 ), "Window0",  
    Color.LightGray );  
// Creating an object Window0, with a size and  
// position, and a background color
```

⁹Operating System

```

XA.AddWindow( Window0 );
// Adding the window to the system
// NOTE: this must be done directly after creating
// the object. Otherwise your program might crash
// due to architectural design

```

10.3 XRectangle

Next we have the XRectangle. This is a hybrid rectangle since it can be used both for showing an image, but also just for viewing an image somewhere on the screen. The default animation is a fading zoom. This is one of the simplest controls we've created, but it renders to be a useful one.

```

Image Cube = Image.FromFile( 'cube.png' );

for( int r = 0; r < 4; r++ ) // Create 4 squares
{
    XRectangle tmp =
        new XRectangle( new Rectangle( (r%2)*100, (r/2)*100, 100, 100 ),
            "Rectangle"+r, Cube );
    tmp.AnimationLength = 10; // A 10 frames animation duration
    tmp.AnimationDelay = 5*r; // Wait for a multitude of 5 frames before
        actually animation
    Window0.AddControl( tmp ); //Adding the actual rectangle
}

```

10.4 XTextBox

This class has the ability to show a text, with a user-defined font, on screen. We've added the ability to show a border around a text, this greatly improves readability, especially when there's a complex background with a lot of transitions. When you change the text, it simply cross-fades from the previous text to the new text.

```

XTextBox Text0 =
new XTextBox( new Rectangle( 100, 100, 400, 60 ), "Text0", "Welkom", new
    Font( "Tw Cen MT", 25.0f, FontStyle.Regular ) );

Text0.BorderColor = Color.Black;
Text0.FillBrush = new SolidBrush( Color.White );
Text0.BorderWidth = 3;

Window0.AddControl( Text0 );

```

10.5 XBrowser

The browser-control was added upon request from Televic to be able to show digital-signage content on a screen. But also for browsing the internet. We've had two implementations of the browser, one using Internet Explorer's renderer and one using google's Chrome.

10.5.1 Internet Explorer

We started of using the well known renderer used in internet explorer. This was the easiest one to use and get results from. Since we're working with the .NET framework, we have an internet-explorer control at our disposal. Normally one would place this control on a windows form, and thus use the browser.

However we can't directly do this, because we do all the drawing ourselves. Therefore we had to create the control in the background, in the constructor of our XBrowser. This was an easy task, however we still didn't have an image of what was rendered. Luckily for us, there's a function `DrawToBitmap()` which returns...a bitmap of the control's state. This did exactly what it should do. But there was a counter-side. It was horribly slow, and even if nothing changed...the drawing would still happen. So it was a successful proof of concept. But there was light at the end of the tunnel! Indeed `DrawToBitmap` is slow, it uses GDI+, which is a hardware independent way of drawing, more about this in the chapter GDI+. Therefore we tried the GDI way of doing things. There's an function called `BitBlt` which allowed us to pass the handle from one device context to another device context. We've made a little test program which had a browser and a picturebox. In the browser we'd watch a youtube-movie, while in the picturebox we'd view the render. We put them side-by-side just to see any differences in speed or delay. The speed difference was enormous! Not only did we have no delays, the processor-usage was ultra-low, spiking somewhere at 5%. But our eyes were deceived. Everything did go that well, but that had a very good explanation. The only thing we really did was, copying the visible part of the browser to an image, however when there's no browser visible (and this is still what we want), no image will be shown. And that was the end of internet explorer's path. We had to start looking for alternatives.



Figure 4: Internet Explorer logo

10.5.2 Google Chrome

Since nearly all the major browsers use Webkit, we had to see what could be done with this piece of toy. Webkit was developed by Apple, to be used in their browser Safari. Webkit is build upon KHTML, a KDE HTML layout engine. In June 2005 whole Webkit became open-source. This was a very good thing because a huge community could improve the speed and rendering capabilities of the engine. Webkit was the first one to pass the Acid3 test where no other engine could even get close to this result. It didn't take long



Figure 5: Chrome Logo

before other companies embedded Webkit in their software. Chrome is a famous browser that uses Webkit for its rendering. At first we couldn't find much information of Webkit and how to embed it.

Since we had a deadline, things needed to go fast, and that's when a very interesting API came by: Berkelium Sharp.

Berkelium Sharp is build upon Berkelium, which is a wrapper around Chromium, which is the actual render-engine for Chrome, which is build upon Webkit. So we have two major keywords here that define our choice: Webkit and .NET. Berkelium was originally used for usage in 3D games. Again we couldn't find much info about the API, but it rather self-explanatory, the author also wrote a webbrowser with it using simple windows forms and using XNA¹⁰, a faming library from Microsoft (more about this topic in the chapter XNA). It proves to be very speedy, whilst not using a lot of resources. When we look at our task-manager we see that another programs starts up with our program: berkelium.exe, I figured that this is where the actual rendering happens, and that the underlying system still uses a BitBlt to do the copying, however that's not something I know for a fact, but it seems plausible.

```
XBrowser Browser0 = new XBrowser(new Rectangle( 50, 50, 640, 480 ) /*the
    location and size of our browser*/, new Size( 640, 480) /*The size of
    the browser-render*/, "Browser0", "http://www.televic.com/" /*The
    Url*/ );

Window0.AddControl( Browser0 );
```

10.6 XButton

Even though we have a program already with some interaction, we can't do very much. Therefore we have created a button. This button (like any other button), can be selected (which is the same as hovering a button with a mouse), can be clicked and has an action attached. By default one could add a NormalState image, and when a user selects the button, a white, translucent border will appear around the button. When you click on a button, the button will briefly switch from selected to normal, and then back again to selected. There's a Clicked-event attached, which can be freely programmed.

```
void initButton()
{
    XButton Button0 = new XButton( Rectangle(800, 600, 100, 100),
        "Button0" ); //Object creation
    Button0.NormalState = Image.FromFile( 'button.png' );
    Button0.BorderWidth = 3;
    Button0.BorderColor = Color.FromArgb(128, Color.White);

    Button0.Tag = "http://www.google.be/"; //An user-defined object can
        be attached to the button

    Button0.Clicked += new XEventHandler( Button_Clicked ); // Subscribing
        the event

    Window0.AddControl( Button0 ); // Adding it to the window
}

void ReturnToRSSFeed_Clicked(XControl sender , RemoteKey e)
```

¹⁰XNA's Not Acronymed (this is not a joke)

```
{
    Browser0.NavigateTo = sender.Tag;
}
```

10.7 XSlideShow

The slide-show control was created because of the demand to browse and show pictures on screen. By default the slide-show can be controlled with the left-arrow and right-arrow buttons on a keyboard, a remote control or of course our cube. You can form a list of pictures from a local repository or from pictures spread over the internet. There's no default animation, you choose which transition mode you want to use e.g. zoom, fading zoom, blocked (pixelated), cube-like and many more. For a long time this was our final control. Sooner than later however we had an urge to create photo-albums and other easy browsing capabilities, which we could do by using buttons... but this would be a lot of work, if not a lot of thinking how this could be done without a lot of bugging. And that's where our super-control saw daylight.

```
XSlideShow ImageViewer = new XSlideShow(new Rectangle(100, 100, 640,
    480), "Slideshow", XSlideShowMode.OpacityZoomMaskRandomize);
ImageViewer.Mask = (Bitmap)Bitmap.FromFile("Mask.png");
//Load a mask, the alpha-channel is used as a mask

List<string> images = new List<string>();

images.Add("Test0.jpg");
images.Add("Test1.jpg");
images.Add("Test2.jpg");

ImageViewer.LoadPictureList(images, false);
// Add images in the picture list, stating that
// the pictures are local (due to parameter 2
// being 'false')

Window0.AddControl( ImageViewer );
```

10.8 XContainer

The XContainer, our final control. Like we said before, we needed an easy to use library-control. We really loved the slideshow control so it didn't take long before we chose to do things the slideshow way, with the nice transitions and so on. Our first-problem was a way to create an image from the controls, because only then we could have our cool transitions (more about this topic in Drawing). In a first implementation you had to make the XContainerPages (which is a descendant of XWindow) and add them to the container so they could be viewed. But this proved to be rather limiting, certainly when we had an infinite number of pages, like when creating an activity calendar, where every day has a different set of activities. That's where the concept of virtual pages was invented. The only thing you have to do is stating how many pages there will be used (where -1 represents infinite). When we go to another page, an event is fired. We send a 'page' parameter, which can be filled in by the program. This greatly limits the size of the program, the resources used and makes it very easy to understand the code, the downside is that things can be a little slower. We still kept the old way of adding pages, but we like to push people to use the virtual pages instead. Another cool thing is that you can put a container in a container (and so on). Giving the ability to create a very lightweight, yet powerful GUI. However it is not advised to nest more than one

level of containers. This would make it rather hard for a user to quickly jump out of a container onto its parent and so on, back to the parenting window.

```
void InitContainer ()
{
    XContainer XCPContainer =
    new XContainer(new Rectangle(175, 150, 700, 500), "Container1",
        XSLideShowMode.CubeMaskRandomize, false /* Is vertical? */, 3 /*
        VirtualPages ? */);

    XCPContainer.VirtualPageRequested += new
        LoadVirtualDelegate(XCPContainer_VirtualPageRequested);
    Window.AddControl(XCPContainer);
}

void XCPContainer_VirtualPageRequested(XContainer sender, ref
    XContainerPage page, int CurrentPage, int TotalPages)
{
    XContainerPage XCP = page;

    XTextBox Text0 =
    new XTextBox(new Rectangle( 100, 100, 400, 60), "Text0", "Page " +
        CurrentPage.ToString(), new Font("Tw Cen MT", 25.0f,
        FontStyle.Regular) );

    Text0.BorderColor = Color.Black;
    Text0.FillBrush = new SolidBrush(Color.White);
    Text0.BorderWidth = 3;

    Window0.AddControl(Text0);
}
```

11 ConnectX



Figure 6: VisualX Logo

ConnectX is our API name for a collection of functions that allows us to connect to the outside world. This API makes it possible to use digital signage inside our program, like browsing pictures, viewing RSS feeds or viewing the daily activities. These classes also enable us to be used in a multi-client environment. In our program we'll use all of these functions to prove the stability.

11.1 ConnectXServer

This class is the base class in a multi-client environment. The server is a simple socket server. The sockets are used as blocking sockets, since these are the easiest to program. Blocking sockets are sockets that halt the thread they run in until a response has been received. The Accept function for instance, keeps on waiting until a client connects to it. This proves to be rather useful because it doesn't use processor-time while waiting, non-blocking sockets however continue to run, and a delay should be programmed. But we want to use every bit of processor time available for useful things, not just waiting. Bound to port 1234 by default. In this implementation we don't use SSL¹¹ or another form of security. Only a plain-text password that needs to match. But since it's plain text, one could easily capture this with a package sniffer. This shouldn't be too big of a problem in a real-life situation where everything is done inside a LAN¹². From the second you connect to the internet, a real form of security should be used, but due to our time-constraint we didn't implement this. When a user connects to the server, an event is fired, allowing to do things on the server side, for instance starting our JointSpace-program. In the background we add this client in our server as a ConnectXEndPoint. The ConnectXEndPoint creates a separate thread per class. This thread checks if the client is still active and if the client has send data. When the client suddenly disconnects, the client will be removed from the system, freeing up resources. When data has been received an event is fired, with a ConnectXEndPoint as parameter, this parameter can directly be used to send a response. Doing so gives us a parallel server-system. This has as advantage that a crashing client doesn't crash the whole program, and a bonus is that it has much faster response times. As disadvantage we can say it's a little bit more complex to program, and it uses a bit more resources.

```
void InitServer ()
{
    ConnectXServer.Port = 8888; //Set the port, the default is 1234
    ConnectXServer.Secret = "Secret"; //Set the secret, both clients need
    the have the same secret
}
```

¹¹Secure Socket Layer

¹²Local Area Network

```

Server = new ConnectXServer(); //Start the server

Server.ClientAdded += new ClientStateChanged(Server_ClientAdded);
Server.DataReceived += new ClientSendData(Server_DataReceived);
Server.ClientRemoved += new ClientStateChanged(Server_ClientRemoved);
}

void Server_ClientAdded(ConnectXEndPoint cxep)
{
    MessageBox.Show(cxep.Name + " has connected!");
}

void Server_ClientRemoved(ConnectXEndPoint cxep)
{
    MessageBox.Show(cxep.Name + " has disconnected!");
}

void Server_DataReceived(ConnectXEndPoint sender, string data)
{
    MessageBox.Show("Data received: " + data); // Here we'll see "Data
        received: Hello Server?"
    sender.SendString("Hello");
}

```

The client is easily implemented with the following code:

```

void initClient()
{
    ConnectXClient.Port = 8888; //Same as the server-port
    ConnectXClient.Secret = "Secret"; //Same as the server-secret

    ConnectXClient Client = new ConnectXClient("Eagle" /*Name*/, "Birds"
        /*Classification*/, "xxx.xxx.xxx.xxx" /*Server-IP*/);
    Client.DataReceived += new
        ReceivedDataFromServer(Client_DataReceived);
    Client.SendString("Hello Server?"); // Send the string "Hello Server?"
}

void Client_DataReceived(ConnectXClient sender, string data)
{
    MessageBox.Show("Server response: " + data); //Here we'll see "Server
        response: Hello"
}

```

11.2 ConnectXDigitalSignageReader

Televic has its own digital signage system¹³. It proves to be a rather useful platform with a lot of extra-features that we can benefit from. We noticed that it's rather easy to add your own applets to this system. With this handy little tool we can access everything from our client-program. We made it possible to read certain news-feeds, query the pictures for a certain user (and/or the public ones) and as a little extra we made it possible to let a user subscribe to a certain activity. The user identity is stored in a public integer called UserID, when we set this to a value not equal to zero, we can query for personal information about the user.

¹³More about Digital Signage in the chapter 'Digital Signage'


```

void InitDigitalSignage ()
{
    ConnectXDigitalSignageReader DigitalSignage = new
        ConnectXDigitalSignageReader("server", "username", "password",
            "location", "location-password"); //Here we make the necessary
            connection to Digital Signage
}

```

All of this is done with the following functions:

Feeds

This function retrieves all the feeds that are registered in the system. It simply returns a list of objects which state what the logo is of the feed, to name of the feed, a description and the actual url to the feed. This object can than be passed to a ConnectXRSSReader. This class does the actual reading, and we even built a nice little feature which tells you what the latest features are (without actual remaking the object).

```

void XCPContainer_VirtualPageRequested(XContainer sender, ref
    XContainerPage page, int CurrentPage, int TotalPages)
{
    XContainerPage XCP = page;
    List<ConnectXRSSFeed> Feeds = DigiSign.Feeds();

    for (int i = 0; i < Feeds.Count - CurrentPage * 6 && i < 6; i++)
    {
        ConnectXRSSFeed feed = Feeds[i + CurrentPage * 6];
        int y = i / 3;
        int x = i % 3;

        XButton FeedButton =
            new XButton(new Rectangle((225 * x), (225 * y), 200, 200),
                "RSSFeed." + feed.Name);

        FeedButton.NormalState = feed.Image;
        FeedButton.BorderWidth = 10;
        FeedButton.BorderColor = Color.FromArgb(128, Color.White);
        FeedButton.AnimationLength = 5;
        FeedButton.Tag = feed;

        FeedButton.Clicked += new XEventHandler(RSSFeed_Clicked);

        XCP.AddControl( FeedButton );

        XTextBox FeedText =
            new XTextBox(new Rectangle((225 * x), (200 + 225 * y), 200, 31),
                "RSSFeed.Label" + feed.Name, feed.Name, new Font("Tw Cen MT",
                    25.0f, FontStyle.Regular));

        FeedText.BorderColor = Color.Black;
        FeedText.FillBrush = new SolidBrush(Color.White);
        FeedText.BorderWidth = 1;
        FeedText.AnimationLength = 5;

        XCP.AddControl( FeedText );
    }
}

```

Images

It enables us to download pictures in a certain given folder (on the server that is). Pictures that don't belong to the user, won't be returned.

```
XSlideShow ImageViewer = new XSlideShow(new Rectangle(150, 150, 630,
    500), "Slideshow", XSlideShowMode.OpacityZoomMaskRandomize, false);

ImageViewer.LoadPictureList( DigiSign.Images("") ); // Insert the
    pictures from a certain folder
```

SubFolders

With this function we can view all the subfolders in a folder, the retrieved folder names can than be passed to the picture folder, which views the pictures.

```
List<ConnectXDigitalSignageFolder> folders =
    DigiSign.SubFolders(PictureFolder);

for (; i < 6 && i < folders.Count - CurrentPage * 6; i++)
{
    int index = CurrentPage * 6;

    XButton AlbumButton = new XButton(new Rectangle((i % 3) * 225, (i /
        3) * 225, 200, 200), "XRectangle");

    AlbumButton.NormalState =
        (Bitmap)Bitmap.FromFile("Images/folder.png");
    AlbumButton.BorderWidth = 10;
    AlbumButton.BorderColor = Color.FromArgb(127, Color.White);
    AlbumButton.Tag = folders[index].Path;

    XTextBox text = new XTextBox(new Rectangle((i % 3) * 225, (i / 3) *
        225, 200, 200), "XTextBox", folders[index].Name, new Font("Arial",
        35.0f));

    text.BorderWidth = 2;
    text.FillBrush = new SolidBrush(Color.White);
    text.BorderColor = Color.Black;

    page.AddControl(AlbumButton);
    page.AddControl(text);
}
```

GetActivities

This function enumerates the activities on a given day, we can easily subscribe to an activity with SubscribeActivity.

GetDinner

Allows us to query for the dinner of the day, this functions returns a string dictionary, with strings as keys. The keys are respectively: "breakfast", "lunch", "snack" and "dinner".

```
void Container_VirtualPageRequested(XContainer sender, ref XContainerPage
    page, int CurrentPage, int TotalPages)
{
    XContainerPage DayPage = page;
```

```

XTextBox Text = new XTextBox(new Rectangle(0, 0, 600, 50), "Text",
    DateTime.Now.AddDays(CurrentPage).ToLongDateString(), new Font("Tw
    Cen MT", 30.0f, FontStyle.Regular));

List<Activity> Activities = DigiSign.GetActivities(
    DateTime.Now.AddDays(CurrentPage) );

if (Activities.Count == 0)
{
    // Tell the end-user nothing is to be shown
}
else
{
    XContainer ActivityContainer =
    new XContainer(new Rectangle(50, 50, 400, 420),
        "ActivityContainer", XSLideShowMode.CubeMaskRandomize, true,
        Activities.Count / 5 + 1);

    ActivityContainer.Tag = Activities;
    ActivityContainer.VirtualPageRequested +=
    new LoadVirtualDelegate( ActivityContainer.VirtualPageRequested );

    DayPage.AddControl( ActivityContainer );
}

List<Object> Dinner = DigiSign.GetDinner(
    DateTime.Now.AddDays(CurrentPage) );

if (Dinner.Count > 0)
{
    XContainer ActivityContainer2 =
    new XContainer(new Rectangle(460, 50, 340, 420),
        "ActivityContainer2", XSLideShowMode.CubeMaskRandomize, true,
        Dinner.Count);

    ActivityContainer2.Tag = Dinner;
    ActivityContainer2.VirtualPageRequested +=
    new LoadVirtualDelegate( ActivityContainer2.VirtualPageRequested
    );
    DayPage.AddControl( ActivityContainer2 );
}
}

```

12 VisualX insight

12.1 DirectFB

DirectFB¹⁴ is a system developed by Denis Oliver Kropp, with whom Philips closely works to develop JointSpace. DirectFB allows us to directly upload a picture to the tv-buffer, and then manipulate it on screen. This is a quit speedy solution, because the image doesn't need to be uploaded everytime it changes its screen location. However there are two problems here: the buffer is limited to 4096 kb, when we work with higher resolutions (1024x768x16bits), we noticed that already 1536 kb was taken by the window only, without even thinking about drawing an image. The second problem is, we do change the picture a lot, to say the least. As resourceful that function might be

¹⁴DirectFB= Direct Frame Buffer

in very few cases, it didn't cut our cake. So we tried a different approach, do all the rendering at the serving-side. That way we have a lot of resources to spend, and a nice centralized place where we can change our program if we wanted to change something.

Below we'll show all the necessary steps needed to make the tv show our content. First of all we need to initialize the directFB library:

```
DirectFBInit( NULL, NULL );
// Initializing the DirectFB library

jslibrc_Init( NULL, NULL );
// Allows us to read the pressed button
// on the remote, and the other way around
```

Next we need to make the actual connection to our tv, subscribe to the window manager and get the window-handle:

```
IDirectFB          *dfb_;
IDirectFBDisplayLayer *layer_;
IDirectFBSurface   *gsurface;
DFBWindowDescription wdesc;

IDirectFBWindow     *gwindow;
DFBSurfaceDescription desc;
IDirectFBEventBuffer *events_;
DFBWindowGeometry   *wingeo =
    new DFBWindowGeometry();
DFBRectangle         winrec;

winrec.x = X;
winrec.y = Y;
winrec.w = Width;
winrec.h = Height;

wingeo->mode = DWGMRECTANGLE;
wingeo->rectangle = winrec;

UpdateScreen = true;

Running = true;
DirectFBSetOption( "remote", "xxx.xxx.xxx.xxx" /* tv ip-address */ );

if( DirectFBCreate( &dfb_ ) ) // returns 0 if everything went according
    to plan
{
    return;
}

dfb_->GetDisplayLayer( dfb_, DLID_PRIMARY, &layer_ ); // Get handle to tv
drawing surface

//Create a window
wdesc.flags = (DFBWindowDescriptionFlags)( DWDESC_POSX | DWDESC_POSY |
    DWDESC_WIDTH | DWDESC_HEIGHT | DWDESC_OPTIONS |
    DWDESC_PIXELFORMAT );
wdesc.posx = 0;
wdesc.posy = 0;
wdesc.width = Width;
wdesc.height = Height;
wdesc.pixelformat = DSPF_RGB16; // Set the color resolution
wdesc.stacking = DWSC_MIDDLE;
wdesc.options = DWOP_NONE;
```

```

layer->CreateWindow(layer , &wdesc , &gwindow);
gwindow->GetSurface(gwindow , &gsurface);

gsurface->Clear( gsurface , 0x00 , 0x00 , 0x00 , 0x00 );
// Clear the surface , should anything still be visible from a previous
// session

gwindow->CreateEventBuffer( gwindow , &events_ ); // Allow us to capture
// buttons

gwindow->SetOpacity( gwindow , 0xFF ); // Making it fully opaque
gwindow->SetDstGeometry( gwindow , wingeo ); // Sets the position on
// screen , possible to overlay
gwindow->SetColorKey( gwindow , 0x00 , 0x00 , 0x00 );
// Defining transparent colors , when a
// color matches this , the underlying visible
// will be visible
gwindow->RequestFocus( gwindow ); //Focus the window

gsurface->Flip( gsurface , NULL /* full screen region */, DSFLIP_NONE );
// Bring the back buffer to front buffer

```

From here on it's just a walk in the park. We can now push data to the screen, without flooding the buffer.

```

unsigned short *bannerPixelBuffer = new unsigned short [r.w*r.h];
unsigned short *bannerPixelBufferAgain = bannerPixelBuffer;
// We make a copy of the pointer , pointing to location 0...

array<int> ^data = Engine->GetBuffer( r.x , r.y , r.w , r.h );

for(int y = r.h-1;y > 0; y-- )
{
    for(int x = 0;x < r.w; x++ )
    {
        Color ^pixelData = Color::FromArgb( data [(y*(r.w))+x] );
        (*bannerPixelBuffer++) =
            R5G6B5VAL((int)pixelData->B,(int)pixelData->G,(int)pixelData->R);
        // copy pixeldata , converting 32 bits colors to 16 bits colors (
        // 565 )
    }
}

surface->Write( surface , &r , bannerPixelBuffer , -r.w*2 );

if( bannerPixelBufferAgain != nullptr )
{
    delete bannerPixelBufferAgain; // we delete the memory, starting by
    // zero ...
}

```

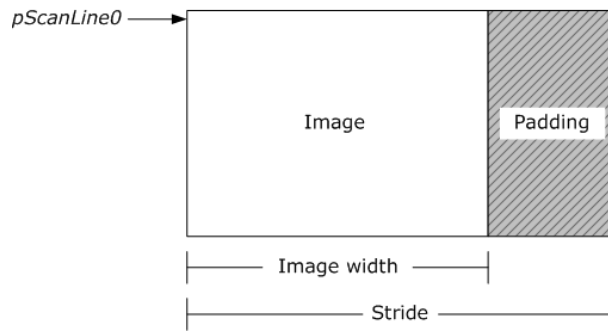


Figure 7: Stride in memory

Like you can see when we make a call to 'surface.Write', we pass the surface, 'surface', we want to draw on, the region, r, we want to draw to and the actual pixeldata. However the last parameter is maybe a little trickier to understand. It's called the stride. The stride or pitch of an image is the amount of bytes it needs to traverse before actually going to the next line of pixels. It's often used for caching purposes. In a 32bits image, the stride is always (imageWidth * 2). This means that the picture has imagewidth padding pixels. It depends on the way an image is drawn to screen if the stride is either negative or positive, when it's negative, this means that the image will be drawn from bottom-up. We also utilise a negative stride, this is because of the fact that RGB memory-bitmaps are usually bottom-up while YUV images are top-down.

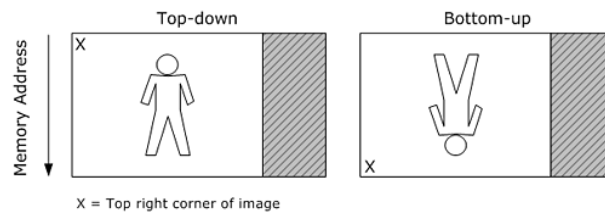


Figure 8: Difference in sign of stride

12.2 Drawing

At the very beginning of our API development, we worked with GDI+. Why? Because it's one of the easiest to use, and easy to understand ways of drawing content directly on a surface. We can do virtually anything we want, no matter how advanced. However, ease of use has one major drawback. We all saw it happen before with other no-brainer API's, some people stop thinking about what they're actually doing. And yes, it happened to me too. Most of the actions (read functions) you take, are not processor intensive tasks. They just happen without you even noticing. For instance creating a rectangle, or an advanced geometrical shape. No the red wire throughout the developing of a graphical library is to make every draw-call count. Drawing to a surface is (with GDI+) the most intensive task around. If you have to much calls to the draw function, well everything becomes a crawl. In our first version we drew every-control on it's own surface, and in a later stage in the program we would draw these sub-surfaces to one big buffer surface. This didn't cause to much trouble when there were only a control or two on screen. But when you make an actual, full-fledged program, we saw a major drop in performance. And how much we twisted and turned, we couldn't find a way around that exact problem. It's a fact GDI+ is slow and new desperate measures needed to be taken...

12.3 A walk through GDI

In the early days (before windows was born), game-developers had a hard time drawing an image. There was no actual standard library to address the video memory, and each company had it's own way of addressing his type of video-card. Yes there was int(errupt) 10, which could draw stuff through the BIOS¹⁵ but it was slow as hell, so they wrote 'drivers' per major video-card manufacturer.

Then windows 1.0 was born, Microsoft's GUI OS. This wasn't a great success and many companies waited to see where this would go. It was not until windows 3.0 that windows created an API called GDI¹⁶ a real standard way of drawing static images to the screen. GDI is still used, up to windows XP it even had some hardware acceleration, but this was dropped in Vista and 7. It could draw to a Device Context (DC), this includes screens but also printers, in a very simple way. It abstracted the DC and the idea of DPI¹⁷. Which contributed to the whole WYSIWYG¹⁸ mindset of Microsoft. However it had a few downfalls, no support for alpha-blending, no vector-drawing and above all a limitation of how many GDI objects could be created. This amount increased from 1200 in windows 98 to 65536 in windows XP and above. Since windows XP, Microsoft introduced GDI+, a wrapper around GDI, but with some extras like anti-aliased 2D graphics, floating point coordinates, gradient shading, more complex path management and support for modern graphics-file formats like JPEG and PNG. However due to the underlying way of drawing, most things needed to be done by the CPU, which halved the speed eight times! The other limitations remained.

12.4 A walk through DirectX

But back in the days, there was another path that could be walked. Like I said before, GDI was only designed for static images, and it wasn't really made to develop games (and yes, after all, we're still creating some sort of a 'game'). That's why nearly everyone stayed with DOS for creating fast-paced graphics. Microsoft wanted to show people that windows was as great for game-development as its predecessor. This was the start of a new era, because WinG was born, an API for creating fast graphics performance applications. This was of course the predecessor of the well known DirectX API, which had DirectDraw as 2D library. Since DirectX 9, DirectDraw wasn't actually present anymore, since most GPUs focus on 3D gaming, DirectDraw is actually done on Direct3D for performance reasons. DirectDraw is since windows 7 succeeded by Direct2D.

Today we do have a choice between many hardware accelerated graphical libraries: Direct3D, Direct2D, XNA and OpenGL. Out of personal experience I didn't even try to go with Direct3D, because of the lines of code before you could actually start drawing something. I even omitted OpenGL, because of the much lower performance gain compared to DirectX and the fact the even though it doesn't require a lot of work to port OpenGL applications to unix-based systems, it will never run as smooth.

12.5 Improving VisualX

In the DirectX arena we have Direct2D and XNA. Direct2D shows us some very promising examples. It was created with only one thought in mind, porting GDI+ to Direct2D.

¹⁵Basic Input Output System

¹⁶Graphical Device Interface

¹⁷Dots Per Inch, the quantity of color-dots per square inch

¹⁸What You See Is What You Get

In the few examples we could find, this looked true. But there's no .NET implementation, and since the whole engine I have right now is written in .NET, I want to use .NET again. So the last one my list: XNA. XNA is an easy to use wrapper around DirectX. An extra advantage is that it can be run on Windows, Zune, Xbox 360 and there's even a Linux port called MonoXNA (which uses OpenGL).

I didn't know very much about this API, only that it was easy, but this scared me because easy mostly means not-versatile. But I was proven wrong, the API was indeed easy to use. However a whole new idea of thinking needed to be used.

The first step is initialization, below is the code used for GDI+ to allow us to start drawing primitives and graphics:

```
Bitmap Buffer = new Bitmap( 1024, 768 );
//Initializing the buffer

Graphics G = Graphics.FromImage( Buffer );
//Initializing the graphics object
```

In the following block of code we'll see that it takes a lot more effort to initialize XNA:

```
PresentationParameters pp = new PresentationParameters();
//These parameters allow us to configure the engine, and where we'll
//see the output
pp.BackBufferFormat = ColorFormat;
pp.BackBufferHeight = BackBufferHeight;
pp.BackBufferWidth = BackBufferWidth;
pp.DeviceWindowHandle = handle; // the handle to the hosting control
// (window, canvas, whatever)
pp.IsFullScreen = false;
pp.PresentationInterval = PresentInterval.Immediate;
GraphicsDevice Device = new GraphicsDevice( GraphicsAdapter.Adapters[0],
GraphicsProfile.Reach, pp );
// The actual device initialization
SpriteBatch spriteBatch = new SpriteBatch( Device );
// This class is where all the rendering will happen
```

The actual drawing code is where the biggest change is visible. When we want to draw an image in GDI+, we type the following line of code:

```
G.DrawImage( test, 0, 0, 100, 100 );
```

What it actually does in the background now is a lot more, first of all it locks the Buffer-image. We do this, so that the image can not be changed while we're drawing, not having to think about this actually speeds up the processing of the image. When the image is locked, the image 'test' is scaled to the size of 100px by 100px. This process is done through the CPU. Afterwards the rescaled image is drawn to the Bitmap and the bits are unlocked again, so another image can be drawn. Again for drawing only a small amounts of images and/or primitives this won't be too much of a problem. But like you can see, the bitmap is being locked per draw-function call. The ease of use is very big, but when we want to draw a lot, this is more like a burden than a blessing. In real game development, the buffer is locked one and then images are pushed upon a pipe-line:

```
spriteBatch.Begin(); //Start of the pipeline

spriteBatch.Draw( testTexture, new Rectangle(0, 0, 100, 100), Color.White
);
//More drawing
//Even more drawing can be done here
```



```
spriteBatch.End(); //End of the pipeline
```

This code is rather self-explanatory, except for the last parameter of the draw function: 'Color.White'. This last parameter can be compared with the effect sun-glasses have when you look through them, where Color.White sunglasses are the regular glasses which pass through all the light and Color.Black would be sleeping mask where all the colors are blocked.

Another major difference with GDI+ is that the spriteBatch only allows to draw images, there is no support for primitives like rectangles or other stuff. Of course these are very useful and we don't wanna lose that! We wrote another wrapper around XNA, to allow us to easily port our application from the GDI+ version, to XNA. This challenged us to re-invent the wheel GDI+ already has, or do we! Luckily for us we don't! We can pass a stream to the Texture2D classes (XNA's equivalent for GDI+'s Image) and we can save an Image to a MemoryStream. In between those two functions we can do the GDI+ rendering. Those functions won't be the most speedy ones around, but they'll still do the job much faster!

12.6 Colors and alpha-channels

Like one could have guessed not everything is as easy as it seems. Since XNA 4.0 their way of blending changed drastically. Before we had the most commonly known way of blending called alpha-blending. In alpha blending the colors of the overlay are mixed with the colors of the underlying plane by a factor 'alpha'.

$$\text{blend}(\text{source}, \text{dest}) = (\text{source.rgb} * \text{source.a}) + (\text{dest.rgb} * (1 - \text{source.a}))$$

Here the alpha-channel can be seen as an invisibility cloak, where a color can be red, but still be invisible at the same time. Nowadays we work with premultiplied alpha blending. Which is a more realistic approach to how the real world works. A windshield has an alpha channel nearing zero, but it doesn't really have a color. The color and the alpha-channel are now dependent values.

$$\text{blend}(\text{source}, \text{dest}) = \text{source.rgb} + (\text{dest.rgb} * (1 - \text{source.a}))$$

INFO: Alpha-channels don't really have a meaning, they're just an extra value attached to an rgb value, which can be used for anything. But it's mainly used for defining opacity.

However we can't just load an GDI+ image (which uses alpha-blending), and hope that some magic will convert the color values to premultiplied alpha blending. In our first attempt to overcome this problem, we took a pointer to the image, and looped over the pixels, converting them to premultiplied on the fly. This did the trick, however, it was slowing down the performance of our program. So we had to be a little bit more creative. So we drew the image into an off-screen `RenderTarget2D` (which is inherited from `Texture2D`) while stating that the alpha-channel for that `RenderTarget` is actually used for alpha-blending. When we now copy the values from our GDI+ image, these will be converted to premultiplied alpha on the fly, through the power of the GPU instead.

12.7 Conclusion

When we compare both API's both in speed and ease of use, we can say that XNA is not only faster, but actually a lot easier to use than GDI+. While writing this I wondered why people would even want to use GDI+. Even the most power-efficient computers can now render by the power of a magnitude faster. We used FRAPS for our benchmarks, both test-systems ran Windows 7 home premium. We noticed that using GDI+ our application couldn't get above 10 FPS¹⁹ on a high-end computer. Running the app on a low power-consuming PC, we noticed a drop to 3 FPS. This is very disturbing when you want to implement animations. I'd like to add that while an animation was running, our CPU was using only one core at max to do everything. So on a quad-core system (or more) the application is at a crawl.

When we ran the same tests while using XNA, we noticed a very big difference of 60 FPS on both systems (the high-end system and the low power-consuming PC). This was only limited due to vertical synchronization (which let's the drawing functions wait until the screen starts drawing from the start again. Also known as the refresh rate), this prevents tearing in the screen at high rates. Disabling this synchronization gave us a speed of 180 FPS!

¹⁹Frames Per Second

13 Televic Signage

13.1 Digital signage in general

Digital signage is electronic display that is able to show information. These displays, can be LCD, LED, plasma,... displays. Most of the time the information that is displayed on the screen, is being controlled on a computer with appropriate software. The software that is used to link the media to the screen can be web-based software, this enables the user to control and manage the screen from every location.

13.2 Televic Signage

Televic signage is a Digital signage system, that is developed in Televic. This system provides a interface were different companies can sign in, were they could manage the locations(sign) in their company. Each company has its own available content that can be easily loaded into the database. The content that can be displayed isn't just plain text, you can also load images, movies, flash, websites,... onto the screen. The manager is also able to make presentation, and schedule them according to the occasion. Before the companies can use this system, they need to buy some credits. These credits are the currency for using Televic signage. These credits will reduce according to the amount you use the system.

13.3 Setup Televic Signage

Before we can deploy Televic signage, PHP, apache and MySQL must be installed on your system. When you are using windows, like we did, WAMP²⁰ offers this in one package. Download the latest version of WAMP and install on the system. The application has been tested and was fully working with: APACHE: 2.2.17, PHP: 5.3.5, MYSQL: 5.0.51a.

Before you can use the Televic signage application, you must enables the required PHP extensions and import some SQL files into the database. The PHP extensions that must be installed are:php_curl, php_gd2, php_gettext, php_imap, php_mbstring, php_mysql, php_mysqli, php_openssl, php_pdo_mysql, php_pdo_sqlite, php_soap, php_sqlite, php_xmlrpc, php_zip.

After you have installed these extensions, you must import Televic_signage_for_branch.sql and Televic_signage_for_branch_empty.sql to PHPmyAdmin. Now you must copy the Televic signage branch(the application) to the WAMP "www" folder. Inside the "config.php" file, which can be found in the include folder, you must adjust the setting according to the server you're using. The application should be running. Note: a detailed "how to install" file can be found in the readme folder of the Televic signage branch.

13.4 Adjustments Televic Signage

In the beginning we made an manager application were we could manage our implementations to the project. Management was rather confusing, because we needed to switch between the Televic manager and our Admin tool. Therefore we integrated our admin tool into the Televic manager.

²⁰WAMP = Windows, Apache, MySQL and PHP

13.4.1 Changing visual experience

Televic signage is used as a media content library. The manager can view the content that is stored in the database, when he is logged in. In the original version of Televic Signage, the content was shown in a long list, with lots of unnecessary information. The picture thumbnail was really small, you couldn't really see which picture you wanted to add to a presentation, without clicking it first. We changed this representation into something more user friendly. Every content is viewed as a large image, with the title of the image. When hovering over a picture, you can tick the picture to add it to a presentation, or delete it from the database. Later on we've added a folder-system, which allows pictures to be in several folders. These folders can have sub-folders and so on. The only restriction we have, is that the full folder path cannot be longer than 1024 characters long.

We've added hovering effects to make it more appealing to the eye. We have integrated these changes throughout the website, so that everything looks uniform. The hovering effect was done through the power of CSS²¹. In the full CSS3 specification we have a few new tricks to use in our applications. We can now specify a transition. This means that going from one CSS-value to another will be tweened²². However it's not actually in the standard yet, and luckily there's only one browser that doesn't implement these non-standard transitions: Internet Explorer. This is truly only for beautification.

```
div.calendar_inactive_weekend
{
    background:white;
    -moz-transition: all 0.5s ease-out; /* for Mozilla */
    -o-transition: all 0.5s ease-out; /* for Opera */
    -webkit-transition: all 0.5s ease-out; /* for webkit (safari, chrome
        and many more) */
    transition: all 0.5s ease-out; /* the official implementation */
}

div.calendar_inactive_weekend:hover
{
    background:gray; /* When we hover now, we'll see that it takes 0.5s
        before the new color is fully visible. */
}
```

Another 'experience' we've added, is the infobox! The infobox is an innovative way of showing the requested information. It pops up from below, but you can't miss it, since it's moving. When you don't hover over the text, the info-bar hides after 2.5 seconds. That way we don't get an annoying rectangle when we try to navigate the site. The hover bar is implemented by javascript.

First we place where we want our infobox to be, and what basic styling it should have (text-color, background, font-size, font-family...) with a required height of zero pixels. Next we define which elements are triggers for the info-bar to show, and with what content.

```
<div onmousemove="ShowInfo('Hello InfoWorld');" >This is some text</div>
```

When a mouse is moved above this control the ShowInfo function is called. Below we see the code that handles the animation. As you can see it's a second degree function, where the closer we get to the end-value, the smaller the steps will be to the next value.

²¹Cascading Style Sheets

²²tweened = the process of generating intermediate frames between two images to give the appearance that the first image evolves smoothly into the second image.

When the mouse is not moved while it's hovering over the element, a watch-dog will trigger and 2.5 seconds later the infobox will be hidden.

```
function ShowInfoBox() {
    ..
    InfoBox = document.getElementById( 'info_box' );
    InfoBoxInner = document.getElementById( 'info_box_inner' );
    wantedHeight = InfoBoxInner.offsetHeight + 40;

    InfoBox.style.height = parseFloat(InfoBox.style.height) + parseFloat(
        ( wantedHeight - parseFloat( InfoBox.style.height ) ) * 0.1 ) +
        'px'; //0.1 is the procentual step-size
    ..
}
```

13.4.2 Tag management

We have integrated the storage of the RFID tags into the Televic MySQL database. Every tag that is stored into the database is assigned to a specific type, a tag can be part of a cube, or it could be a tag of the type "gallery" which would open your personal media folder. Every tag is also given an personId, this way we can link tags to a person, thus linking a cube or object to persons.

13.4.3 Person management

In our system we have the idea to give every person his own cube. With this cube the user can identify himself to the system. Because we wanted this process to be straightforward we added a person management page. On this page an administrator could add a new person into the system, and also see a list of the current users that are added to the system. On this list the administrator can easily see which users are assigned to a cube and which don't. This is shown by either a cube(assigned) or a warning sign(not assigned). When a cube is not assigned, a click on the persons picture will add a cube The administrator must then follow a certain amount of steps, involving the interaction of the cube, to assign the person to the specific cube. When hovering over a persons picture, some options will be shown. Clicking on the red circle will remove a person from the system, and the cube that is assigned. Every person has been assigned his own media folder, clicking on this folder will enable you to add pictures into the folder of this specific person. When the users want to view his picture on the end-user application, he will see his own pictures from his personal folder and the pictures available to everyone, he will not be able to watch picture assigned to other persons.

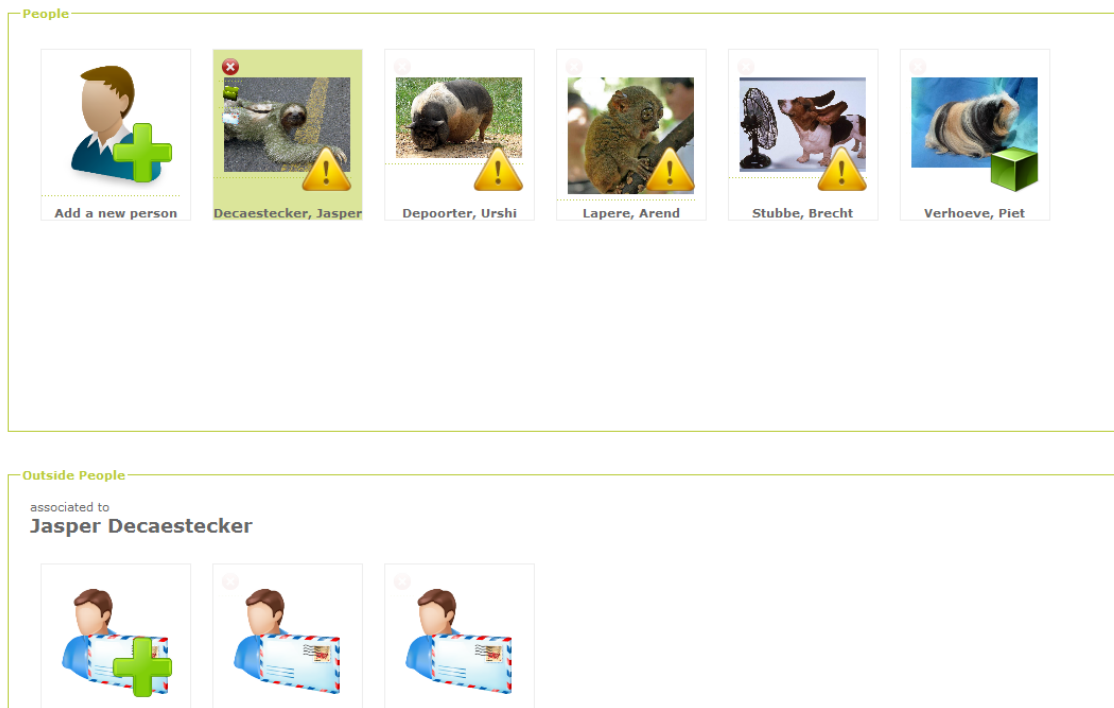


Figure 9: view of people management page

13.4.4 Plugin management

When we want to add a new feature to the end-user application, it is possible to write a plugin using our plugin system. We wanted the plugins to be easy to deploy, so we added a plugin manager. Before you can upload the plugin to the manager, there are some requirements on how to format you're plugin. The plugin that must be uploaded must be zipped and the main directory must contain:

- an index.php
- a logo.png
- a nfo.txt

The only real necessity is the index.php, because the plugin will not work without a proper direction to the plugin code. the logo.png is the picture that will be displayed in the plugin management page. the nfo.txt file will be displayed as extra information when hovering over a plugin. A demo file is available on the CD.

Management - Plugins

Plugin List



Information for testplugin:

```
:: INFO
title: TestPlugin
author: Me
description: This plugin is a test
```

Figure 10: view of the plugins page

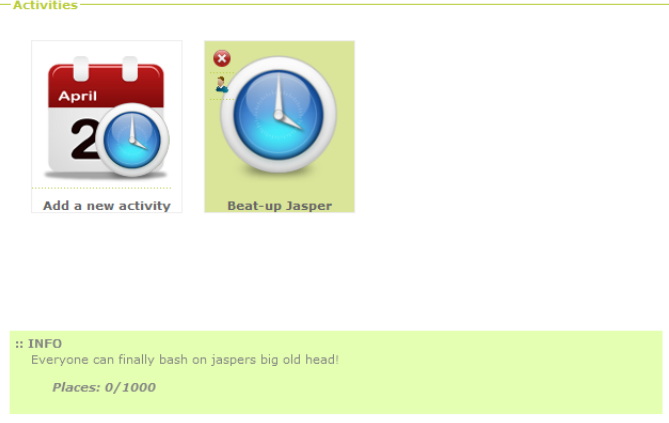
13.4.5 Activities

On this page the administrator can add/view activities on each day. When hovering over an activity, the activity can be removed, or you can view the list of people who subscribed. When an activity is selected a short description is shown underneath the selection. The people can subscribe to these events when they go to the calendar on the TV screen. Besides the activities that are shown on the calendar, you can also view the menu for each day. The administrator can use Televic Signage to change the menu of each day.

Calendar

M	T	W	T	F	S	S
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3

Activities



Information for Beat-up Jasper:

```
:: INFO
Everyone can finally bash on jaspers big old head!
Places: 0/1000
```

Menu

Breakfast:

Lunch:

Snack:

Subscribers

First name	Last name	Picture
------------	-----------	---------

Figure 11: view of calendar page

13.4.6 RSS feeds

On this page RSS²³ feeds can be added into the database. These RSS feeds can then be viewed on our application that is running on the tv. as in the other sections, we also implemented the same layout of the page. when hovering over the RSS picture, you have the option to remove or edit the RSS feed.

13.4.7 RSS feeds historical & technical

RSS was designed by Dave Winer at UserLand software in 1997. It was first used in 1999 by netscape. Since 2001 they dropped support for it. So Winer continued the development inside his company UserLand. The basic usage for RSS-feeds (as they are called) is to allow an end-user to quickly see what has changed on a website, without actually looking at every website to see what has actually changed (or not). This task can be done with a feed-manager that keeps track of the possible changes that have happened. The creation is often automated, but can easily be written by hand.

```
<?xml version="1.0" ?>
<rss version="2.0">
  <channel>
    <title>Students are cool</title>
    <link>http://www.google.com/</link>
    <description>BLA BLA BLA</description>
    <item>
      <title>Article one</title>
      <link>url</link>
      <description>this is an article</description>
    </item>
  </channel>
</rss>
```

This can be interpreted by any XML-parser, like the one in .NET. This way we only need to know the url to the feed, retrieve the data and let it parse. The layout is almost always the same with every feed.

13.5 The Code behind

13.5.1 Showing content

First we make a fieldset to include our pictures, the layout of this fieldset is adjustable in the CSS file. Then we create a title for our fieldset using the legend tags. A div tag is added, to make sure the images are centered inside the fieldset.

Now we will explain the changes made in the PHP code. First we draw a image to the screen, each item represents an item, the first item enables users to add a new person,rss feed, ... This image is encapsulated in a div with class"rss_feeds", this takes care of the layout of the image. Note that our class is named "rss_feeds", but this class is used for all content that is shown on this way, the person management and plugin page use the same class. A link is added to the picture, which will bring users to a new page when pressed. A text is placed under the image, aligned in the center.

After we added an "new item" button, we show the content that is available in the database. we request all data from a specific table and order it by name. For each row we have found, with each row being an separate item, we will draw a picture to the screen. The procedure is the same as the new item button, but the picture itself and its

²³Really Simple Syndication

name is extracted from the database. With each picture we add at least 1 button, this small button is placed upon the image itself. The button is used to delete the selected content from the database. Another button can be added to show additional information of the content. These buttons are placed inside a div with class "rss_delete", where the CSS code will take care of the layout.

```
<fieldset class="fieldset_datagrid"
  style="height:400px;position:relative;">
  <legend>RSS Feeds</legend>
  <div style="bottom:20px; right:20px; left:20px;
    top:30px;overflow-y:auto;position:absolute;">
    <?php
      echo '<div class="rss_feed">';
      echo '<a href="index.php?page=new_rssfeed"></a><br/>';
      echo '<b
        style="position:absolute;bottom:0px;text-align:center;
        width:100%;">Add a new RSS Feed</b>';
      echo '</div>';

      $query = mysql_query("SELECT * FROM rss_feeds ORDER BY
        title",$connection);
      while( $array = mysql_fetch_array( $query ) )
      {
        echo '<div class="rss_feed">';
        echo '<br/>';
        echo '<b
          style="position:absolute;bottom:0px;text-align:center;
          width:100%;">'.stripslashes( $array[ 'title ' ] ). '</b>';
        echo '<div class="rss_delete"><a
          href="index.php?page=rssfeeds&delete_item
          =' . $array[ 'id ' ] . '"></a></div>';
        echo '<div class="rss_delete" style="top:32px"><a
          href="'.stripslashes( $array[ 'link ' ] ). '"></a></div>';
        echo '</div>';
      }
    ?>
  </div>
</fieldset>
```

13.5.2 Deleting content

When the delete button is pressed, which is shown when hovering over an item, this item will be deleted from the database. The delete button is a link to its own page, with a Get variable which contains the id of this item. When this Get variable is set, we send a mysql query, where the item with the id of the selected item will be deleted.

```

if ( isset ( $_GET["delete_item"] ) )
{
    $query = "DELETE FROM rss_feeds WHERE id='{$_GET["delete_item"]}'";
    mysql_query($query, $connection);
}

```

13.5.3 Zip

Because we want our plugins to be easy to deploy, we use the zip extension available in PHP. With help of this library, we could install these plugins by uploading a zip file to our server.

When the zip file is submitted, we copy the zip file from our computer to the server. When the file is successfully uploaded, the user is notified. hereafter we will extract the uploaded zip file to a specific path, this can easily be done with the php zip library. after the extraction we delete the zip file. Now we have a folder with the name of the plugin, "installed" on the server. This plugin can now be used on our tv application.

```

if(isset($_POST["submit"]))
{
    $target_path = "../plugins/";
    $target_path = $target_path . basename(
        $_FILES['file_upload']['name']);

    // upload file to server
    if(move_uploaded_file($_FILES['file_upload']['tmp_name'],
        $target_path))
    {
        echo "The file " . basename( $_FILES['file_upload']['name']) .
            " has been uploaded";
    }
    else
    {
        echo "There was an error uploading the file , please try again!";
    }
    $name= $_FILES['file_upload']['name'];
    $name = substr($name, 0, strrpos($name,"."));

    // extract zip to server
    $zip = new ZipArchive;
    if ($zip->open($target_path) == TRUE)
    {
        $target_path = "../plugins/".$name."/";
        $zip->extractTo($target_path);
        $zip->close();
    }

    // delete zip
    $target_path = "../plugins/" . basename(
        $_FILES['file_upload']['name']);
    unlink($target_path);
}

```

13.5.4 Appbrowser

This code will find all plugins which are installed and encode them to JSON.

We open the directory were the plugins are installed. For each plugin, we will get the content from the nfo file which should be included in each plugin. The content inside

the nfo file is the description of the plugin. We place a list of all plugins inside an array. This array is then encoded to JSON, ready to be read by our C# application, which will show this content on our Television application.

```

if ($handle = opendir( '../plugins/' ))
{
    while (false !== ($file = readdir($handle)))
    {
        if ($file != "." && $file != "..")
        {
            $description =
                file_get_contents( '../plugins/' . $file . '/nfo.txt' );
            $description = nl2br($description);
            $description = str_replace("\n", "", $description);
            $description = str_replace("\r", "", $description);
            $description =
                substr($description, strpos($description, "description"));

            $data[$i]->Name = $file;
            $data[$i]->Description = $description;
            $data[$i]->ImageUrl =
                'http://' . $_SERVER['SERVER_ADDR'] . '/branch/plugins/' .
                $file . '/logo.png';
            $data[$i]->ApplicationUrl =
                'http://' . $_SERVER['SERVER_ADDR'] .
                '/branch/plugins/' . $file . '/';
            $i++;
        }
    }
}
closedir($handle);
echo json_encode($data);

```

14 Plugin System

This Plugin system enables us to make applications for our system, without the need to edit the code of our application. The C# application is then able to load these plugins and show the application on the screen. The plugins are written in PHP. The communication between PHP and C# is made possible with JSON. There are 4 different Controls that can be made with the plugin:

- Container: Used for making pages
- Textbox: Used for displaying text to the screen
- Button: Draws a button on the screen
- Rectangle: Can be used for layout

In the code below we will show you an example how to add a button to the screen. First we make a new ConnectXAppControl. Then we select the type of the control, in our case this control will be of the type Button. Note that order of the parameters are not of importance. We will enable the button to accept events, else we would not be able to select it and click it. We then define the width and height of the button, we can also position the button everywhere in 2D space by changing the X and Y parameter. The last parameter will change the background color of the button.

```

$data [$m] = new ConnectXAppControl();
$data [$m]->Type = Button;
$data [$m]->AcceptEvents = true;
$data [$m]->Name = "button1";
$data [$m]->Width = 200;
$data [$m]->Height = 150;
$data [$m]->X = 200 ;
$data [$m]->Y = 0;
$data [$m]->BackColor = hexdec("6066FF33");

```

A list of all the parameters are available in the "VisualX.php" file.

14.0.5 TV guide

The first application we made for the new plugin system was a TV guide. This application shows every tv show of the Flemish tv post. We did this by parsing the information we get from the zita.be Tv guide. This information is then translated to the controls we can make with the plugin system. For every Tv channel we have, we will create a button, under these button a textbox will be placed. If we select a channel, we draw a rectangle for every show, and populate these with some text, this text is of course the name of the show. When we created all the pages with the controls, we encode this data to JSON. This data is then been read by the C# application and is shown to the tv screen.



Figure 12: Main menu of the Tv Guide

15 How everything works?

When you first start this program, you'll see the following screen. This means that you should place a personal cube. From the moment you place your cube, you are logged in. You get a warm welcome message, together with the owners' name.



Figure 13: Start of the program



Figure 14: Main menu

When we click OK with our cube, we enter the selected application, which in our case is the album browser. With this album browser we can view the pictures that are available on the digital signage side. We have browsers and pictures. When clicked on a folder, we will see the content of that folder which can be folders or pictures and so on.



Figure 15: Album



Figure 16: Picture browser

Back on the main menu we click on the calendar icon. We enter the activity browser. Here we can view which activities and what food will be served each day. To navigate you can use the arrows on the cube to go either left (backwards in time) or right (forwards in time). We have a list of activities, where you can subscribe. A green tick means you're subscribed for that activity. A limited amount of participants are available. When there's no room left, no option will be available to subscribe.

When we click on a picture, a slideshow appears and we can get a closer look at the picture. With the left and right arrows we can browse through all the pictures that are in that particular folder hallo. We return to the album menu by clicking the top-right button twice. Once to return to the album browser, and the second time to return to home.



Figure 17: Activity manager



Figure 18: Feed browser

When we select the newsfeeds on the main menu we enter the feed browser. Here we see all the registered feeds. Again, navigating can be done with the arrows on the cube. In the background an actual RSS feed will be read, allowing us to see the header, and a small introduction text.

When we click a feed, you'll see a summary of all the messages. There's a nice transition between each message. If there's a picture available, it will be shown. When we click on an interesting message, a browser will open, allowing us to read the full story.



Figure 19: Browsing through the articles!



Figure 20: Application browser

Next we have the application browser. Here we see all the registered plugins. These plugins are installed in digital signage and allow to easily expand the usage of the program.

Take for instance testplugin, this plugin gives the tv-magazine. Where we can view what program is currently playing on specific channel. When clicked, a more in-detail view of that particular channel can be browsed.



Figure 21: A plugin at work

16 The Future

If this product would be commercialy used, there are still some issues that need to be fixed. First of all the televic signage applications needs to be adjusted. Our addition to the application is a neccesity to demonstrate our application, this however made the application a little more complex and in some cases contradictory to the original Televic signage application. Though it will need some thorough thinking to make the application more user friendly.

Security was never one of our concerns here, nor was it a part of our assignment (although we did try to implement some solutions). However should this be used on a larger scale security measures should be taken in account for. Like using HTTPS to access the digital signage for instance, an SSL connection for ConnectX instead of unsecured sockets.

Although we've written our programs to be as scalable as possible, this wasn't a direct request. So it hasn't been tested to see what the results would be. In a future version this is something that should be tested.

17 Field trips

17.1 Philips Brugge

Because it was necessary to acquire some information about Jointspace, Televic arranged a meeting with Jean-Marc Harvengt, who is one of the founders of Jointspace. Our main concern was the compatibility of our application on a real Television. We didn't really know how well our application would be running on the real hardware, because we were using a simulator provided by Philips. Everything turned out great, the application was performing as well on the television as it did on the simulator. Some questions about speed performance and the future innovations of JointSpace was cleared up.

17.2 IBBT Gent

The IBBT Gent were the first to come up with the idea to make a smart platform for use in nursing homes,... Televic and IBBT worked together to realise this project. Because of this cooperation, we were invited in IBBT to explain the work we have put in our project. The meeting was a fairly a technical presentation, as it purpose was to transfer the knowledge we have received, to be used in the "Woonzorgcentrum De Vijvers".

18 Complications

18.1 Compiling libnfc

Before we could use the libnfc library we needed to compile the source code to object code, that can be used on our PC. While browsing the libnfc community page we quickly learned how to compile this code. We tried compiling the code with the Mingw compiler, but the code that was generated didnt work really well when we started programming. We tried recompiling and fiddling with the settings, Although without any success whatsoever.

We tried compiling the code on an other PC. This was a success, the generated code

could be used without the random errors we have gotten on our previous tries. The possible problem was the target OS. We first compiled it on a 64bit system while the second time, the OS was a 32bit system. Although we are not sure that this caused the problem, because the code should be 64bit compatible, at least if we believe the people from the libnfc community.

18.2 Televic Signage

In the beginning we had problems setting up the Televic signage on our local machines. The code we have gotten, was mainly adjusted to run in the cloud. Most settings could be quickly adjusted, however some parts of the code needed rewriting to get it working on our local machines.

Besides code problems, the documentation that was written to setup the system was outdated. In this manual you could find a list of extensions that must be enabled to get the signage going, however because of this outdated document, some extensions that were required, were not mentioned. It was really more like a try and catch approach to get the application running. We were still lucky to have Ivo Verdonck, a student who had worked with Televic signage, by our side. Without the help of Ivo, we would have required much more time to get the system operational.

18.3 The power of green computing

When we opened the box of these energy efficient eeebox PC from asus, we were dazzled by the promising power these computer offered. When we got the system running however, we were rather disappointed, the hardware looks good on paper, but working with these computer was like going back into time. We tested our application on this PC and the performance was really bad. We quickly discovered that our bottleneck was the CPU. With GDI we would not use the GPU for drawing, XNA was the solution to this. With XNA we use the GPU for drawing, The CPU is not bothered any more with drawing tasks, which were to much to handle for this rather low end CPU.

Conclusion, on these energy efficient systems, it is important to use every resource available!

18.4 Direct FB

18.4.1 Drawing images with alpha channel made glitches

In our first attempt to use jointspace to draw only a portion of the screen we used PNG files to test the abilities of the library. Why? Because it has some extra features like semi-transparency. If we can show PNG files, it's safe to say that we could draw anything. However sending alpha data with the write function gave some strange effects like non-transparent parts and other parts fully transparent. To solve this problem we looped through all the pixels in the image, and drew a rectangle with a size of 1 by 1 pixel, with the color supplied by the image pixel. This gave us the wanted result, however too much overhead to be actually usable. So we started the development of a graphics library on top.

18.4.2 Connection cannot be closed

Another problem we've noticed is the inability to close a connection with a television during a program-session. To actually close a connection we'd need to shut down the

program, and restart it. If we don't do this, a memory leak is present at the tv-side, causing the television to actually crash after starting the program a third time. However due to the browser crashing on close, this solution can not be used at the time. This is not necessary in a setting where only one computer servers one tv, however a setting with one computer serving multiple tv's would want to have as much resources available to serve the actually connected users.

18.4.3 Connection with tv broken

When a television is shut down during a session, this will not render an error, instead the program will keep on running, in the background, using up resources. The only way we can think of to solve this, is to ping to the television to see if it's still turned on. This, however, doesn't solve the problem when a user changes channels.

18.4.4 Enumerating televisions

There's already a build in feature for enumerating tv's. However we, as win32 developers, don't have this tool at our disposal. Therefor we needed to write that function ourselves. To do so we captured the broadcast packages, and hard-coded that in our program. This may not be a good solution in the long run, it gives us change to quickly implement this feature. This kind of hack should be possible to transpose to the "Connection cannot be closed"-bug.

18.4.5 Browser crash

For some strange reason we can't actually put our finger on, we get a crash whenever the browser is or was visible during a session. We only get a generic error from windows that something went wrong, but nothing more. This makes it harder to debug, but we managed to narrow it down to the drawing functions. This is not something we've managed to fix thus far.

19 Televic Demo day

Each year Televic organizes a demo day, where all the students can demonstrate their project. The message must be handed over in under 5 minutes. The purpose of this day is to bring the students closer and give some information to the employees. At the end of the demonstrations there was time for a nibble and some drinks. Of course there was some tension, because the 3 best performances could win a price. We ended second, each winning an Ipod Touch.



Figure 22: There was much rejoice !!!

20 Conclusion

We've come to the conclusion that jointspace, with all the benefits it may have, has a growing process ahead. In our current project we used some 'code hacks' to get to a certain point. For the RFID part we are rather satisfied. Although no out-of-the box classes are provided by .NET (java does have an out-of-the box solution for reading smartcards and such); we can say it was fairly duable to implement this ourselves. The choice not to work with the provided service from alcatel, is solely based upon the fact that we don't want to be dependent of alcatel.

For the total project we can say we've had a great success in several fields. First of all we accomplished everything that was asked, and a lot more. We both had a fantastic experience here at Televic. The project we have chosen was right down our alley, we really enjoyed learning these new things.

It was fun to be able to use 6 different programming languages in our project. Sometimes it was awkward to combine these different languages, many a time we were programming PHP in our Visual studio IDE ... but we managed our way through. This internship was a great preparation for the real world. The description of the project was broad, but this allowed us to make our own decision in how we would like to use these technologies. There was a common goal to be met, but we had the freedom to choose which path we would walk.

We sure hope that the work we have put into making our application will be used in future projects. It would really be fantastic to see our application or even a tiny percentage of it, being used in nursing and care homes.

21 Bibliography

21.1 Information links

<http://www.libnfc.org/documentation/introduction>

http://en.wikipedia.org/wiki/Radio-frequency_identification

<http://www.technovelgy.com/ct/Technology-Article.asp?ArtNum=1>

<http://jointspace.sourceforge.net/>

<http://www.php.net/>

[http://msdn.microsoft.com/en-us/library/aa473780\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa473780(v=vs.85).aspx)

<http://www.xnadevelopment.com/>

21.2 Software

Visual studio 2010 Professional

Mikitex

Lyx

Notepad++

MinGw

Cmake

WampServer

SQL Server Management Studio