city cloud

BSc Project IN3405
August 4, 2011

Delft University of Technology
Faculty of EEMCS

Johan Laanstra - 1509268
Jos Kraaijeveld - 1509225
Tom Verhoeff - 1308394

Exam Committee
ir. B.R. Sodoyer
dr. Phil. H.G. Gross
M.J. Vermeulen

SUMMARY

The City Cloud is the project made by team Olife, the Dutch finalists for the Microsoft Imagine Cup 2011. The Imagine Cup is an annual student software design competition in which over 300.000 people participate every year. The finals were held in New York this year.

The project is centered around Open Data, and creating an environment in which sharing and using data is accepted and easy. We designed and implemented a *data broker* system, which is able to different data sources in different formats and make them accessible through one API.

We have accomplished this through creating an OData web service, with a system to dynamically add data sources through ‘connectors’. To accommodate developers who try to create applications with the available data, we also built an easy way for them to publish HTML5/Javascript applications to every mobile platform. Their applications, so-called ‘building blocks’, will be loaded dynamically into our main City Cloud mobile application so there is no installation required by the end user.

Although we have not won the Imagine Cup contest, we are still excited about what we built and the way we built it. The process has cost a lot of time but has definitely been worth all the effort.

PREFACE

This document is the final report for the City Cloud project, a project for the course *IN3405 – Bachelorproject*, which is a mandatory course in the bachelor curriculum of Computer Science at the Delft University of Technology. The project was carried out as part of the Dutch student team that represented the Netherlands at the worldwide finals of Microsoft's Imagine Cup 2011 in New York. We hope this report will give the reader the much required insight in both the technical aspect and the process we have gone through to reach the final product.

We would like to thank the following people for their support and contributions:

- Maarten-Jan Vermeulen (Microsoft) for being our principal at Microsoft and a never-ending source of motivation.
- Brian Hewitt (Microsoft) for arranging meetings and facilitating us with a room to work in when necessary.
- Felienne Hermans (TU Delft) for being our supervisor during the project and supporting us with her valuable Imagine Cup experience.
- Tiago Espinha (TU Delft) for providing feedback on both our Imagine Cup and technical challenges.
- Arie van Deursen (TU Delft) for being the supervising professor and supporting us throughout the project.
- Jochem Toolenaar (Rotterdam University) for developing the City Cloud concept and presenting the idea in New York.
- Oana Nitu (TU Eindhoven) for creating the business case for the City Cloud and presenting it in New York.
- Marcel van der Burg (CapGemini) for providing feedback throughout the project and opening up CapGemini resources when necessary.
- Eric Lawrence (Fiddler) for creating the best network snooping tool available, it assisted us a lot during development.

Jos Kraaijeveld, Johan Laanstra and Tom Verhoeff

Delft, August 2011

GLOSSARY

API. *Application Programming Interface*, a means to communicate with a given system from outside of that system.

AtomPub. A publication-level protocol for publishing and editing resources on the web.

Building Block. In our system, a HTML5/Javascript application meant for mobile devices, which will be accessible from the City Cloud.

Connector. A piece of software to connect external third party data sources to the City Cloud.

CMS. *Content Management System,* a system used by websites to make publishing new content easy without having to edit the underlying website.

Data. (Digital) information which can be used in applications.

Dependency Injection. A software design pattern in which a system's dependencies are loaded during runtime.

Framework. The underlying, functional part of the system, unrelated from user interfaces.

Imagine Cup. Student software competition held by Microsoft, annually.

JSON. *JavaScript Object Notation,* a data-interchange format based on name-value pairs.

Open Data. The notion that data is not hidden but publically available for everyone to use.

Team Foundation Server. A centralized source code control system with additional features, like gated check-In, server-side analysis and many more features.

TABLE OF CONTENTS

1 INTRODUCTION

In this chapter we will introduce our project and the rest of the report. Our project is different from most bachelor projects because we are part of the Imagine Cup. Therefore we will first introduce the Imagine Cup. After that we will introduce the vision behind our Imagine Cup concept called "City Cloud" and in the last section of this chapter we will point out the structure for the rest of this report.

1.1 IMAGINE CUP

The Imagine Cup is an annual worldwide technology competition organized by Microsoft. Students all over the world are challenged to use technology to "solve the world's toughest problems". The Imagine Cup was organized for the first time in 2003 and has since then grown to one of the biggest student competitions around the world with over 350.000 students participating. The competition features multiple categories of which Software Design and Embedded Development are the biggest. Some of these have a local preliminary while others have an online competition with multiple rounds. Each year the finalists in all categories and awards are brought together during the worldwide finals held on a major location somewhere around the world.

In the Netherlands the focus for Imagine Cup has always been on Software Design and Game Design. We had multiple teams attending the worldwide finals during the last five years, but only one team ever made it to the second final round. Johan and Tom were involved with the Dutch finalist team last year and noticed their concept was okay but lacked real technical challenges. Further research showed the same thing happened the year before. Both teams consisted of business-oriented students who won the Dutch local finals with a good concept and presentation, but lacked the technical background to build something that really impressed at the worldwide finals.

This year we wanted to learn from earlier experiences and form a team that brings together conceptual, business and technical expertise. To achieve this we formed a team together with Microsoft. Together with students from The Eindhoven University of Technology and the Rotterdam University of Applied Sciences we developed a concept, business case and implementation. The design and implementation of the actual software serves as our bachelor project and is the main subject of this report.

1.2 CITY CLOUD

The project we have been working on is called the "City Cloud". The vision behind this is best explained by our team member Jochem Toolenaar:

"As more and more information processing capabilities are embedded throughout the environment the city is becoming smarter. These new Ubiquitous Computing devices have the ability to sense and act upon what they are sensing. And as more and more of these "smart objects" connect to the internet they create an "Internet of Things".

With these new developments we will see a huge rise in location based data, not only from these new "smart objects", but also from other sensors placed throughout the city. And social media and other mobile apps also create a huge amount of data.

If we want to advance technologically as an Urban society, all this data needs to be easily accessible for third parties. If all this data is openly available and easy to access, it creates a new set of tools for innovators. The data could be used in new mobile applications or city hard/software. The problem however is that a lot of data is still inaccessible. And using different sources of data is troublesome because of the lack in standards.

With the City Cloud we want to solve these problems. By using the powers of cloud computing like flexible scalability and low costs. We want to create an environment where developers can easily share data and easily use data. With this we want bring down the barrier of using live location based data, so that developers, scientists and artists can focus on creating technology that will help contribute in creating a smarter city."

This is the vision behind the system we have been working on.

1.3 REPORT

In this report we will describe what we did to make the City Cloud come to life. The report is structured as follows. We start with the analysis in Chapter 2. It covers the thoughts behind the system, including requirements and actors. Chapter 3 offers an introduction to some of the technologies we used and functions as a background for people who are not familiar with these technologies. Chapter 4 covers our design phase and changes we made to the design during implementation are covered in Chapter 5. We explain our testing process and other ways to ensure quality in Chapter 6. The organization of the project is covered in chapter 7. We end this report with our conclusion in Chapter 8 and some recommendations for further development of the system in Chapter 9.

2 ANALYSIS

2.1 PROBLEM DEFINITION

Open Data initiatives are starting to increase in numbers, offering developers the opportunity to use or re-use data which had been unavailable before. There are, however, a few issues left unaddressed in the current Open Data movement. Firstly, the amount of publically available information is still very limited. In the Netherlands, the largest repository is Overheid.nl[1], which only has 61 different datasets. To top it off, all these datasets are limited to static data and could very well be outdated. Secondly, the data available at different repositories and even within repositories differs in format, has different extensions and varying structuring. The lack of uniformity proves a barrier for developers trying to work with the available data. This is because the developers need to learn new technologies per data source, as well as reformat data when trying to combine different sources.

Governmental (or public) data isn't the only type of data which can be relevant, however. It can increase in value enormously if combined with, for instance, data generated through social media. A good observation is that somehow most of all data is linked to a specific location. This leads to an elegant way of solving the given problems. The location attribute serves as an easy filter: it ensures relevant results and prevents 'overwhelming'.

[1] http://register.data.overheid.nl/package

To address the Open Data problems and further the advance of the Open Data movement, it is necessary to have a system which serves as a *data broker*. The described issues can be solved by making a system which has access to all the different data sources, is able to request and translate data from said sources and provide a clear interface towards developers. To further facilitate developers, there needs to be an easy way for developers to publish their location-based applications to the masses through smartphones.

The system to be developed is called 'City Cloud' and consists of the following subsystems:

- A web service with a clear API, which can access different data source endpoints by translating incoming queries and ensures the output format is uniform.
- A web application to provide upcoming developers with the tools required to use the web service and display available data.
- A mobile application which serves as a publication platform for applications and displaying relevant information from the web service based on a user's location.

2.2 REQUIREMENTS

To provide proper requirements, it is necessary to identify the different actors involved with the system. In the System Design Document we have identified four types of actors: end users, developers, data providers and system maintainers. The requirements for each actor type based on the MoSCoW-model (Coley Consulting, 2011) can be found in the System Design Document as well. In this section we provide a short summary per type.

End users must be able to use their smartphones and the web application to visualize the data made available through the system based on a specific location. They must also be able to access applications submitted to the system through their smartphone.

Developers must be able to access the data service and filter the available data through querying. The must also be able to publish their application through the system.

Data providers must be able to make their already existing data sets available through our system through a connection. They must retain full ownership of their data.

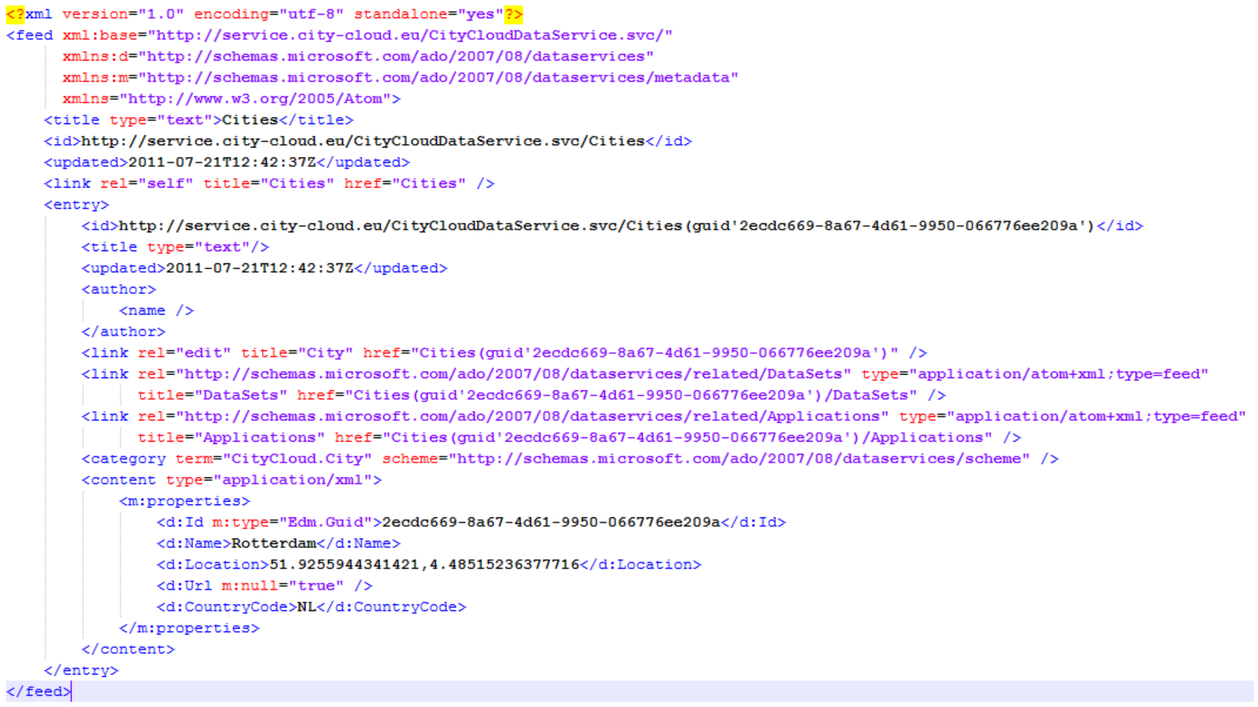
The system maintainers should be able to monitor the system and be able to remove unwanted applications or data sources.

3 TECHNOLOGIES

This chapter functions as an introduction into some of the technologies we used for the City Cloud. It can be used by someone who is not familiar with these technologies to get a quick look at what the technology is about. Some of these technologies were also covered in our initial research document, but the focus was not on the different technologies at that point.

3.1 ODATA AND WCF DATA SERVICES

OData[2] is a web protocol developed by Microsoft and published under the Open Specification Promise[3]. It provides a standard way to query and update data and a way to unlock data stored in silos in today's applications. It builds on existing web technologies such as HTTP, AtomPub and JSON. OData makes deep commitment to URI's to identify resources. A typical OData feed is shown below.



```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://service.city-cloud.eu/CityCloudDataService.svc/"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://www.w3.org/2005/Atom">
    <title type="text">Cities</title>
    <id>http://service.city-cloud.eu/CityCloudDataService.svc/Cities</id>
    <updated>2011-07-21T12:42:37Z</updated>
    <link rel="self" title="Cities" href="Cities" />
    <entry>
        <id>http://service.city-cloud.eu/CityCloudDataService.svc/Cities(guid'2ecdc669-8a67-4d61-9950-066776ee209a')</id>
        <title type="text"/>
        <updated>2011-07-21T12:42:37Z</updated>
        <author>
            <name />
        </author>
        <link rel="edit" title="City" href="Cities(guid'2ecdc669-8a67-4d61-9950-066776ee209a')" />
        <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/DataSets" type="application/atom+xml;type=feed"
              title="DataSets" href="Cities(guid'2ecdc669-8a67-4d61-9950-066776ee209a')/DataSets" />
        <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Applications" type="application/atom+xml;type=feed"
              title="Applications" href="Cities(guid'2ecdc669-8a67-4d61-9950-066776ee209a')/Applications" />
        <category term="CityCloud.City" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
        <content type="application/xml">
            <m:properties>
                <d:Id m:type="Edm.Guid">2ecdc669-8a67-4d61-9950-066776ee209a</d:Id>
                <d:Name>Rotterdam</d:Name>
                <d:Location>51.9255944341421,4.48515236377716</d:Location>
                <d:Url m:null="true" />
                <d:CountryCode>NL</d:CountryCode>
            </m:properties>
        </content>
    </entry>
</feed>
```

Figure 1. A typical OData feed.

WCF Data Services[4] is Microsoft's implementation of the OData protocol. It is included with the .NET Framework 4.0. The version included with the .NET Framework currently supports OData v2. OData v3 is currently in development including a new version of Data Services with v3 support.

3.2 WINDOWS AZURE

Windows Azure[5] is the Cloud platform of Microsoft used to build and host web applications. It was announced at the Professional Developers Conference in 2008. Also an initial preview was released at that moment. Windows Azure is a Platform as a Service (PaaS) solution running in the Microsoft Cloud. Together with Microsoft's Software as a Service (SaaS) solution it is part of Microsoft's Cloud platform. All application in Azure run on the Windows Azure Runtime which is basically .NET extended with some Azure specific functionality.

[2] http://www.odata.org/
[3] http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx
[4] http://msdn.microsoft.com/en-us/data/bb931106
[5] http://www.microsoft.com/windowsazure/

Windows Azure has three components: Compute, Storage and Fabric. Compute offers the computational capabilities of Azure. Compute includes Web and Worker roles. A web role is used for hosting of web applications, while a worker role is used for background processing. Storage offers scalable storage solution in the form of Blob, Table and Queue storage. You can read more on the storage capabilities in our research document. Fabric consists of the physical underpinning of the platform. Compute and Storage components are part of the Fabric. Fabric offers Access Control Services, Caching capabilities and the Service Bus.

3.3 LINQ

LINQ[6] stands for Language Integrated Query. It is the part of the .NET Framework that adds native querying features to .NET. LINQ defines a set of method names, called standard query operators and standard sequence operators, along with translation rules to translate the expression into an expression using the method names. Instead of translating these expressions to method names, an expression can also be translated to for example SQL. This offers unlimited possibilities because you can translate an expression into any other query language.

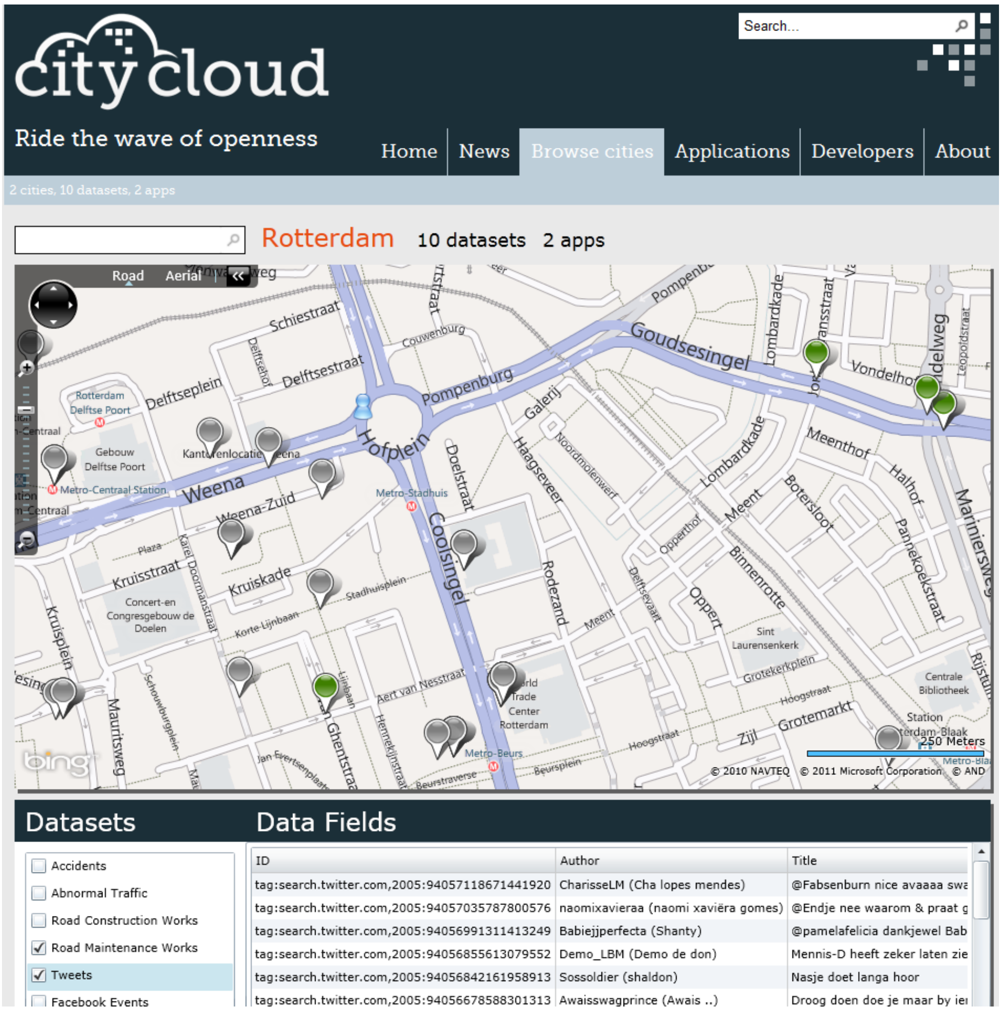
LINQ was inspired by SQL and functional programming languages. A lot of the theories behind LINQ are also used in functional programming languages.

3.4 SILVERLIGHT

Microsoft Silverlight[7] is an application framework for developing rich internet applications. Its run-time environment is a plug-in for most modern web browsers, but can also run on desktop versions of Windows and the mobile platform Windows Phone 7. Silverlight uses a subset of the .NET Framework for programming code behind, while user interfaces are declared in Extensible Application Markup Language[8] (XAML).

The City Cloud project uses Silverlight for the general example application on the website. This demo application lets users combine and plot data onto a map based on the specified location. Below is a screenshot of the website containing the mentioned Silverlight application.

[6] http://msdn.microsoft.com/en-us/netframework/aa904594
[7] http://www.microsoft.com/silverlight/
[8] http://msdn.microsoft.com/en-us/library/cc189054(v=vs.95).aspx



☐ Accidents
☐ Abnormal Traffic
☐ Road Construction Works
☑ Road Maintenance Works
☑ Tweets
☐ Facebook Events

ID	Author	Title
tag:search.twitter.com,2005:94057118671441920	CharisseLM (Cha lopes mendes)	@Fabsenburn nice avaaaa swa
tag:search.twitter.com,2005:94057035787800576	naomixavieraa (naomi xaviëra gomes)	@Endje nee waarom & praat g
tag:search.twitter.com,2005:94056991311413249	Babiejjperfecta (Shanty)	@pamelafelicia dankjewel Bab
tag:search.twitter.com,2005:94056855613079552	Demo_LBM (Demo de don)	Mennis-D heeft zeker laten zie
tag:search.twitter.com,2005:94056842161958913	Sossoldier (shaldon)	Nasje doet langa hoor
tag:search.twitter.com,2005:94056678588301313	Awaisswagprince (Awais ..)	Droog doen doe je maar by iei

Figure 2. A screenshot of the City Cloud web application.

3.5 WINDOWS PHONE

Windows Phone 7[9] (WP7) is Microsoft's new mobile smartphone platform. It was announced in February 2010 and the first phones featuring the WP7 OS have been released in October/November 2010. Microsoft provides 2 frameworks to develop applications for the new Windows phones. Silverlight for Windows Phone and XNA[10], both based on the .NET framework. XNA is mainly used for games, so in our project we will only use Silverlight for Windows Phone.

Within the City Cloud we will use the WP7 application to provide the end-user with a native mobile application experience that integrates tightly with the WP7 OS. It will serve as a portal towards all applications available in the City Cloud.

[9] http://www.microsoft.com/windowsphone/en-us/default.aspx
[10] http://msdn.microsoft.com/en-us/aa937791.aspx

3.6 HTML5 AND DATAJS

HTML5[11] is the fifth revision of the HTML standard that was originally created in 1990. Since HTML4 was standardized in 1997 the internet has been evaluating, new features and standards were introduced not only through new standards, but also by web browsers, various plugins and common practice of combining technologies. HTML5 is still under development and aims to bring all these different developments together in one standard. It has a huge focus on multimedia and location-aware experiences.

HTML5 also focuses on the mobile web experience that has grown in importance with the broad availability of smartphones. All major smartphone platform including iPhone, Android and Windows Phone 7 support HTML5 in their browsers. On mobile devices HTML5 provides a richer user experience and improved usability. Before HTML5 was introduced proprietary device and browser APIs were necessary. Especially HTML5's uniform geolocation support renders it very useful for mobile applications.

The City Cloud has a huge focus on making data and application built on this data available to as much people as possible. Providing a framework that natively supports HTML5 makes this possible by making sure one application runs on all major smartphone platforms. Within the City Cloud we call those applications "Building Blocks".

datajs[12] is described by its developers as a JavaScript library for data-centric web applications. It is a cross-browser library. Modern protocols such as JSON and OData can be leveraged through this library. It also enables a developer to use browser storage mechanisms.

The combination of HTML5 and datajs enables a developer to easily develop portable applications using the City Cloud data service. As of now, there is one application developed as an example: the Health Emergency application. This application looks at the user's current location passed to it by the (mobile) browser and displays the nearest hospitals and automated external defibrillators (AEDs) on a map.

3.7 WINDOWS IDENTITY FOUNDATION

Windows Identity Foundation[13] (WIF) is Microsoft's solution for identity management. It tries to externalize identity logic from the core of an application. It can be used to build claims-aware and federation capable applications by using security token and user attributes.

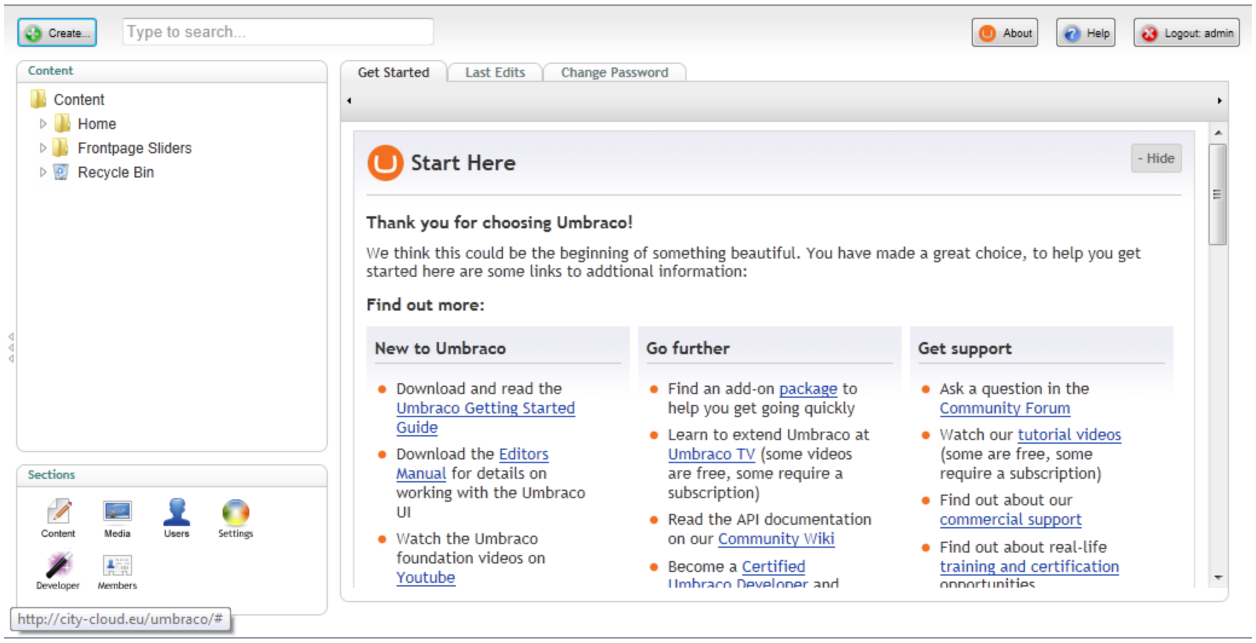
WIF is mainly used for authentication purposes. It integrates nicely with existing authentication mechanisms. Besides authentication WIF also offers authorization capabilities based on user attributes.

[11] http://dev.w3.org/html5/spec/Overview.html
[12] http://datajs.codeplex.com/
[13] http://msdn.microsoft.com/en-us/security/aa570351

3.8 UMBRACO

Umbraco[14] is an open source Content Management System (CMS) built in ASP.NET. The Umbraco project is coordinated by the Umbraco HQ, an privately owned company. Umbraco is one of the most deployed Content Management Systems on the Miscrosoft stack with over 110.000 active installations. Umbraco HQ makes sure project stays up to date with new cutting edge technologies.

Umbraco is a CMS for developers. It takes some time to figure out all the details, but when set up it works great.



Start Here

Thank you for choosing Umbraco!

We think this could be the beginning of something beautiful. You have made a great choice, to help you get started here are some links to addtional information:

Find out more:

New to Umbraco

- Download and read the Umbraco Getting Started Guide
- Download the Editors Manual for details on working with the Umbraco UI
- Watch the Umbraco foundation videos on Youtube

Go further

- Find an add-on package to help you get going quickly
- Learn to extend Umbraco at Umbraco TV (some videos are free, some require a subscription)
- Read the API documentation on our Community Wiki
- Become a Certified Umbraco Developer and

Get support

- Ask a question in the Community Forum
- Watch our tutorial videos (some are free, some require a subscription)
- Find out about our commercial support
- Find out about real-life training and certification opportunities

Figure 3. A screenshot of the Umbraco control panel.

3.9 REACTIVE EXTENSIONS

Reactive extensions (Rx)[15] is a library provided by Microsoft that makes it possible to use certain reactive programming principles within C# .NET applications. It allows for asynchronous data streams to be represented as observables automatically propagating changes to the data. These data streams can also be queried using LINQ which was described in section 3.3.

Within our system Rx comes in very handy when designing the mobile application that relies heavily on an external data source (our own OData service in this case). Mobile data connections have a high variation in speed and therefore making the application wait for a transaction to finish is not an option. Using the Rx framework it is possible to query the data service on a background thread and automatically propagate incoming data to the UI when it becomes available. This can be done without interrupting the UI and therefore providing the best end-user experience.

[14] http://umbraco.com/
[15] http://msdn.microsoft.com/en-us/devlabs/gg577609

3.10 CODE CONTRACTS

Code Contracts[16] is Microsoft's implementation of the Design by Contract design principle. By using design by contract you can define preconditions, postconditions and class invariants. In that way assumptions and specifications can be formally specified in the source code. These contract function as checked documentation. Code Contracts was originally a research project conducted by Microsoft Research and developed as part of the Spec# language. Later it was integrated in the .NET Framework 4.0. Design by contract was invented by Bertrand Meyer in connection with the design of the Eiffel programming language. How we used Code Contracts will be explained in section 6.1.2.

4 DESIGN

During the design phase the system was divided into three parts: the framework, the mobile platform and the web application. The focus was on the design of the framework, because of its important role in the overall system. The mobile platform and the web application were seen as two additional applications that use of the framework. In this chapter we will cover our design choices. Where relevant user interface design will also be covered.

4.1 FRAMEWORK

The original design of the framework can be found in Appendix B under System Architecture. During development some changes were made to the data model. Also the subsystem decomposition changed a little bit. Changes in the different parts of the original design of the framework will be explained in the following paragraphs.

[16] http://msdn.microsoft.com/en-us/devlabs/dd491992

4.2 DATAMODEL

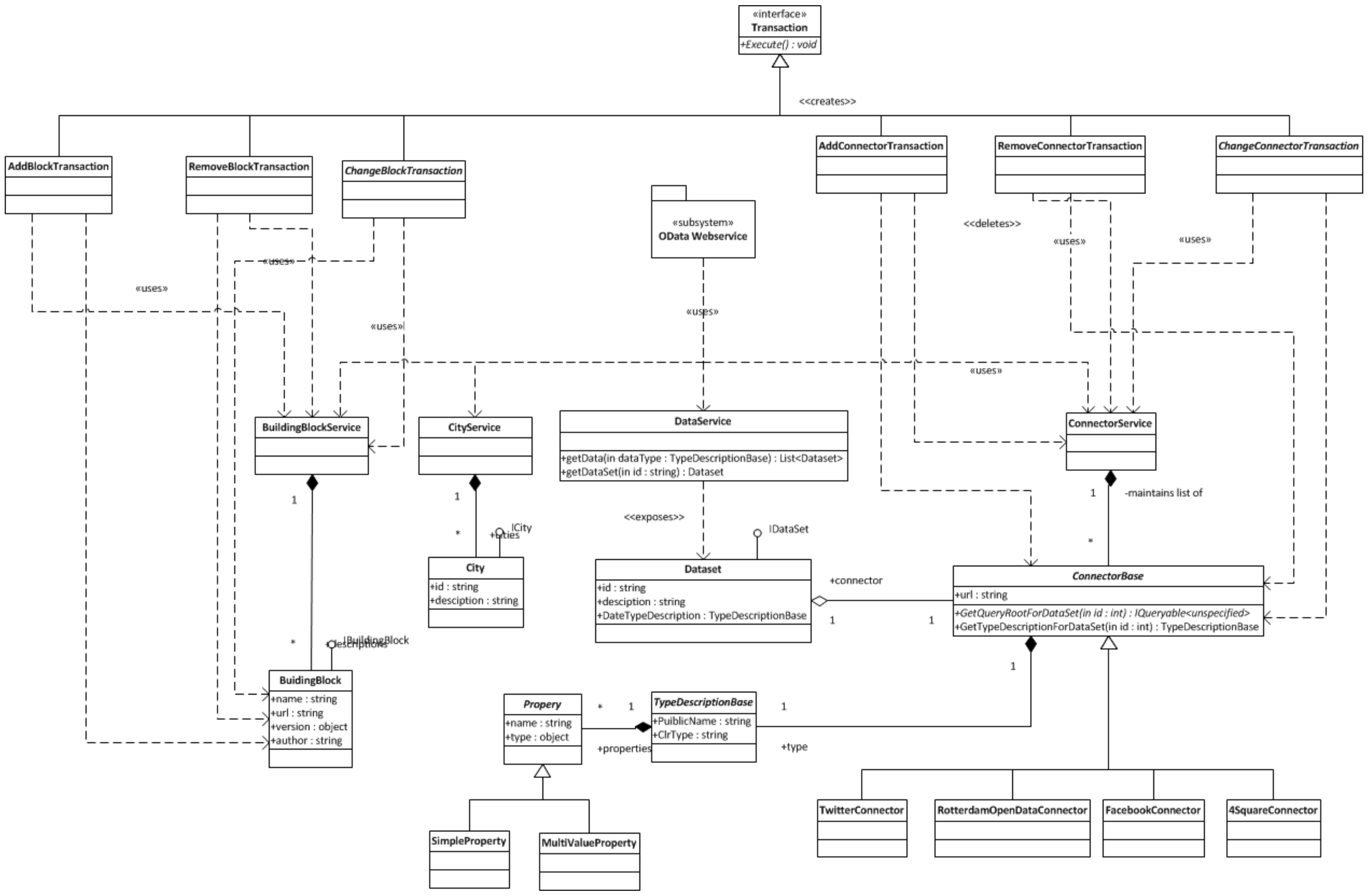


Figure 4. The final datamodel of the City Cloud framework.

We changed the way we handled new datatypes compared to the old design, because we didn't like the way we had originally designed it. It became too complicated and it didn't work out the way we wanted. Instead of letting developers do a lot of work to define a new datatype as originally intended we let developers add new .NET classes and use reflection to find the properties of a class. In addition a developer only needs to provide a short description indicating which properties should be exposed and which not. This offers lots of new possibilities.

The connector part was also changed to reflect changes in the datatype handling. It made the connector part much simpler, because it could now work with built-in LINQ query providers out of the box. This also made the developer experience much better.

At the end of the development we felt the need for a CityService. By using the CityService developers would be able to find datasets and apps relevant to that city and it helped the display of apps and datasets on the website. This was just a small addition to the model and not a lot of changes were required.

4.3 SUBSYSTEM DECOMPOSITION

During development two subsystems were added. The first one is the BuildingBlockHandler which handles specific httprequests for building blocks. The second one is the ScreenshotHandler which handles requests for screenshots of building blocks. Together with the OData webservice they form the entrypoints of the framework.

- OData webservice: Entrypoint for retrieving data from the system, including data on available building blocks and cities.
- BuildingBlockHandler: Entrypoint for loading building blocks from the system into the mobile application. A building block can be identified by its URL.
- ScreenshotHandler: Entrypoint for retrieving screenshots of applications and building blocks. A screenshot can be identified by its URL.

4.4 EXTENSIBILITY

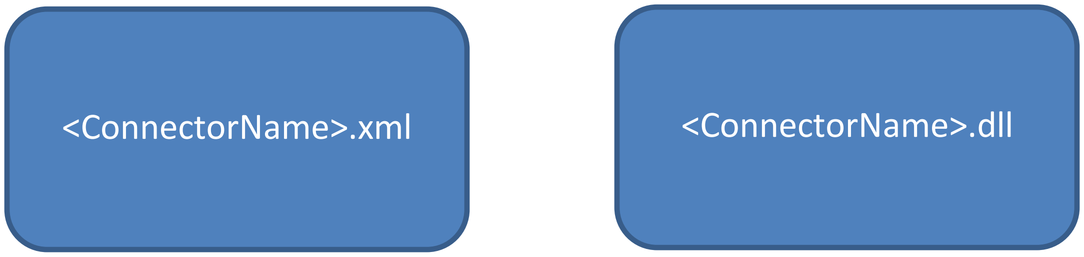
The power of the City Cloud lies in its extensibility and its power to support every useful data format. In this section, we describe how the system can be expanded.

4.4.1 CONNECTORS

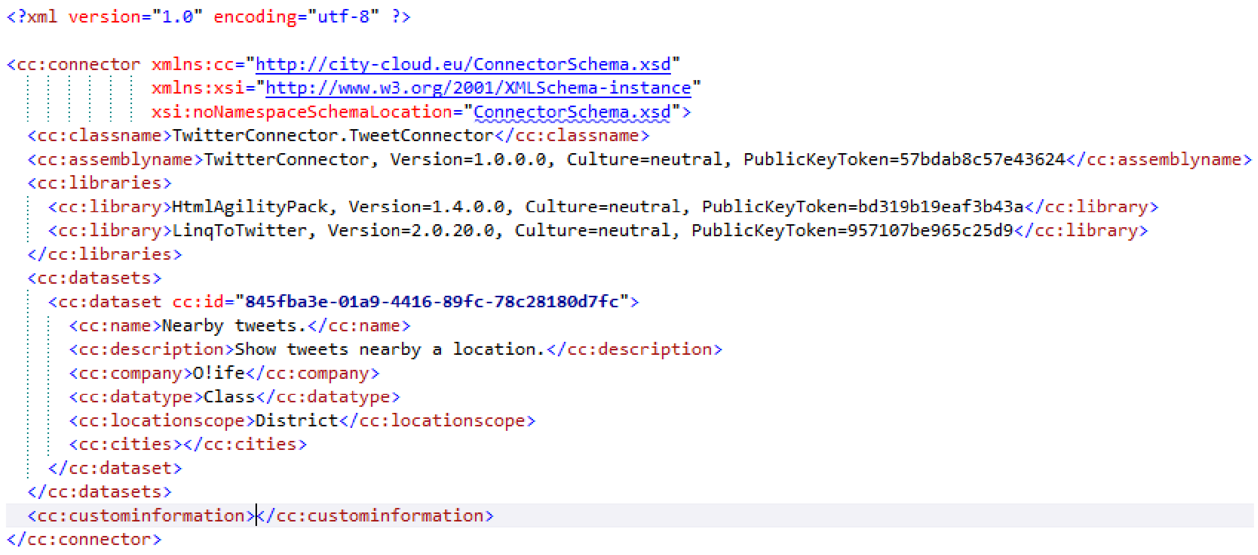
Important to note is that the City Cloud in its form and idea does not store data itself. All the data made publically available is done so through the Connector system. A connector is a piece of software connecting an external data source to the City Cloud and is required to do two things:

1. Retrieve the data requested by a query from the system. This sometimes means the query has to be translated into a query accepted by the third party data source.
2. Ensure the return data is in the correct format. This is any data structure implementing IQueryable within the .NET Framework.

To add a connector to the system there are two things needed: A connector description document, which is an .xml file and an assembly containing the required classes, which is a .dll file.



The programmer writing a connector needs to specify the data types and data sets the connector supplies. The data types are created by providing the system with a class inheriting and implementing TypeDescriptionBase, a class our developer library provides. This class should be in the connector assembly. The specification of datasets is done through the connector description document, and specifies remaining descriptive factors of a dataset, like the scope in which the data supplied is relevant. Below is an example of such an XML file.



```xml
<?xml version="1.0" encoding="utf-8" ?>

<cc:connector xmlns:cc="http://city-cloud.eu/ConnectorSchema.xsd"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="ConnectorSchema.xsd">
  <cc:classname>TwitterConnector.TweetConnector</cc:classname>
  <cc:assemblyname>TwitterConnector, Version=1.0.0.0, Culture=neutral, PublicKeyToken=57bdab8c57e43624</cc:assemblyname>
  <cc:libraries>
    <cc:library>HtmlAgilityPack, Version=1.4.0.0, Culture=neutral, PublicKeyToken=bd319b19eaf3b43a</cc:library>
    <cc:library>LinqToTwitter, Version=2.0.20.0, Culture=neutral, PublicKeyToken=957107be965c25d9</cc:library>
  </cc:libraries>
  <cc:datasets>
    <cc:dataset cc:id="845fba3e-01a9-4416-89fc-78c28180d7fc">
      <cc:name>Nearby tweets.</cc:name>
      <cc:description>Show tweets nearby a location.</cc:description>
      <cc:company>O!ife</cc:company>
      <cc:datatype>Class</cc:datatype>
      <cc:locationscope>District</cc:locationscope>
      <cc:cities></cc:cities>
    </cc:dataset>
  </cc:datasets>
  <cc:custominformation></cc:custominformation>
</cc:connector>
```

Figure 5. An example of a connector .xml file.

You can see the three standard xml document type declarations followed by the connector tag. It further contains a classname which should exist in the assembly specified in the assmblyname tag. Our twitter connector has dependencies on some libraries which are specified with their assembly-fully-qualified-name in the libraries tag. After the libraries a list of datasets follows. The id specified for a dataset is internally used to identify a dataset and to get the query point for the dataset. In the custominformation tag developers can include information about other required information for the connector to work, like API keys or logins to a specific service. The connector system allows us to gradually expand the available data in a clean way, while still keeping it relatively easy for third parties.

4.4.2 BUILDING BLOCKS

Building blocks are an integral part of bringing the City Cloud to end users. With the rise of HTML5 and smartphones, we thought it possible to provide an easy way for application developers to publish their work on all mobile operating systems at once. The idea is as follows: we provide native wrapper applications for every platform which can dynamically load HTML5 / Javascript applications uploaded to the City Cloud. This wrapper application will be accustomed to the user's current position to provide a tight link with the City.

A building block is such an HTML5 application. Like connectors, they are described by an XML file to indicate where the application is relevant and such can be deployed anywhere without having to separately install something on an end users mobile device. By providing a connector to give new data, an existing application can increase in value as well. As an example: we have created an application to show locations of nearby AEDs in the Netherlands. If someone supplied the locations of AEDs in Korea through a connector, the same application could be used there instantly.

4.5 PORTALS

4.5.1 WEBSITE

Designing a website is structurally different from designing the backend of a piece of software. The most important aspect is its layout and the way it comes across to the user. It has be clear what information is displayed, where different information is found and it should fit with the corporate image you are trying to form or underline. An early decision was to mainly facilitate developers with the website, without disregarding the general public. This lead to the notion that the latest added datasets and latest added applications should be visible clearly. Below is an early draft of the website design.

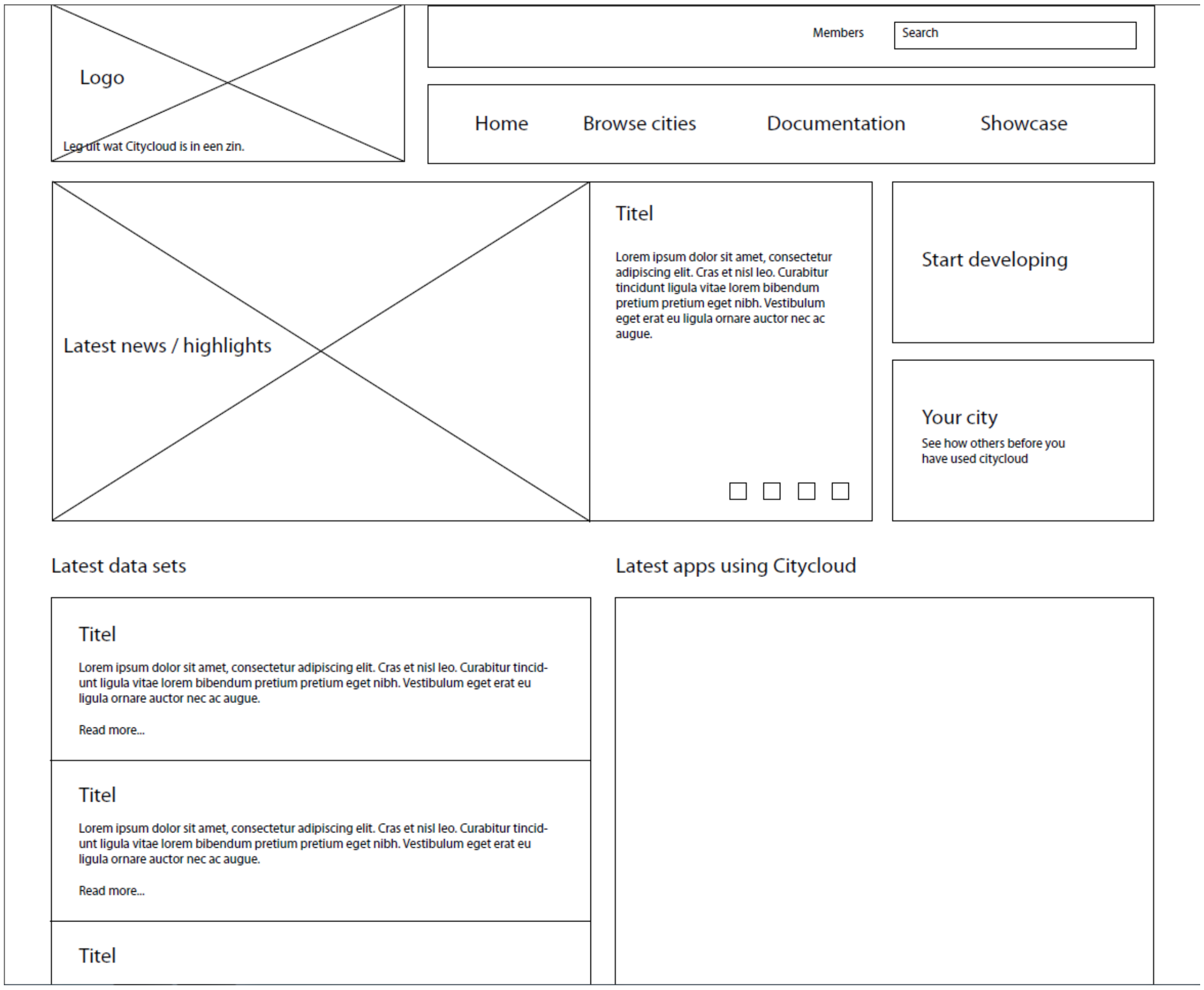

Logo
Leg uit wat Citycloud is in een zin.
Members
Search
Home
Browse cities
Documentation
Showcase
Latest news / highlights
Titel
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras et nisl leo. Curabitur tincidunt ligula vitae lorem bibendum pretium pretium eget nibh. Vestibulum eget erat eu ligula ornare auctor nec ac augue.
Start developing
Your city
See how others before you have used citycloud
Latest data sets
Latest apps using Citycloud
Titel
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras et nisl leo. Curabitur tincidunt ligula vitae lorem bibendum pretium pretium eget nibh. Vestibulum eget erat eu ligula ornare auctor nec ac augue.
Read more...
Titel
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras et nisl leo. Curabitur tincidunt ligula vitae lorem bibendum pretium pretium eget nibh. Vestibulum eget erat eu ligula ornare auctor nec ac augue.
Read more...
Titel

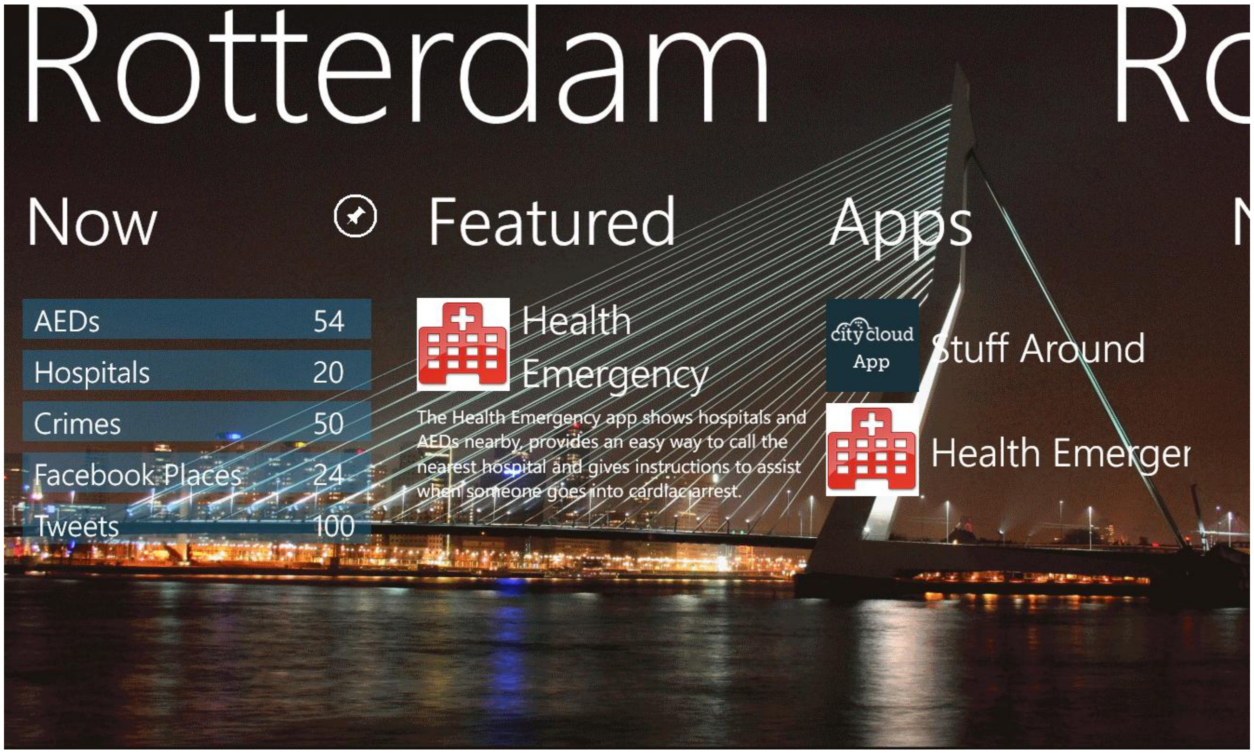
Figure 6. The initial design for the City Cloud website.

An important aspect of the website was the web application to showcase the available data. This Silverlight application has been designed according to the Model View ViewModel (MVVM) architectural pattern, which is a branch of the popular Model-view-controller (MVC) pattern. It relies heavily on the notion of data binding, which is the concept of binding underlying models to graphical elements. The MVVM pattern was first introduced by Martin Fowler (Fowler, 2004). This leads to two easily distinguishable subsystems in the web application: Utility classes and ViewModel classes.

4.5.2 PHONE APPLICATION

When designing a mobile phone application fitting in with a platforms native experience is one of the key focus points. This holds especially for Windows Phone 7 which has a strong focus on integrated experiences. The key aspects of this design have been clearly identified by Microsoft in the UI Design and Interaction Guide for Windows Phone 7 (Microsoft, 2010).

One of the core components of the platform is the hub experience. A hub brings together all related content about a certain subject. Examples of hubs already available in the OS are the People Hub, Pictures Hub, Music hub, etc. Bringing together all related content about a City is exactly what we want to achieve with our mobile application and therefore the application was designed as being a City Hub. Within this hub all data and applications about a city and from different sources can come together. This is what the City Hub for Rotterdam could look like:



AEDs	54
Hospitals	20
Crimes	50
Facebook Places	24
Tweets	100

The Health Emergency app shows hospitals and AEDs nearby, provides an easy way to call the nearest hospital and gives instructions to assist when someone goes into cardiac arrest.

Figure 7. A screenshot of the mobile wrapper application.

Just like the web application the mobile applications for Windows Phone 7 are built using Silverlight. Therefore the mobile application is also designed with the MVVM architectural pattern in mind. MVVM ensures the application is both testable and maintainable without having to rely on UI interaction. Combining MVVM with the Rx framework described in section 3.9 makes sure the applications UI remains smooth while data processing is going on in the background.

5 IMPLEMENTATION

The actual implementation was the most time-consuming part of our project. Basically the challenge was to get as much functionality implemented before the Imagine Cup worldwide finals. Agile development fits in perfectly, although due to the complexity of the system some designing had to be done upfront. More on our implementation plan can be found in the System Design Document that's available as Appendix C. In this chapter we will discuss our experiences during the execution of our implementation plan. The focus will be on things we had to do differently and why we had to do them differently.

5.1 FRAMEWORK

Implementation started with the framework, specifically the OData Webservice. This order of implementation gave us something to build on. The OData service we had designed contained some tricks, which is why we could not use the default implementation included in the .NET Framework. The main one is the fact we had to support all types of data, even the types we do not know of right now. One of the challenges in the first two weeks was how we could use dependency injection with WCF Data Services and ASP.NET. Some good examples were found on the internet and development went on. A big challenge occurred due to a conflict between the way OData works and the way we would like it to work. The problem is that we had a collection of type DataSet and each type DataSet had a collection of a different type (the data). The issue is the fact each DataSet had a collection of a different type. OData doesn't support this way of nested collection, so we needed to find a workaround. We decided to go with the solution of decoupling the collection containing the data from the collection that contains the DataSet object. In this way the DataSet collection contained all the information about a dataset and each dataset had an additional collection containing the data. With this solution we lost the references between the actual data collection and the dataset collection, but it left enough possibilities for the system to work with.

After the first two weeks development moved on to the connector part. As explained above some changes were made in this part of the design, but no big challenges occurred during the development of this part of the framework. A challenge occurred when we tried to actually build a connector ourselves. We tried to connect Twitter to the city cloud and when trying to authenticate a problem occurred we did not foresee. Twitter uses OAuth. OAuth is meant to allow third parties access to a resource without giving the third party access to the user's credentials. In OAuth there are normally three parties: the user, the third party and the resource. In the Twitter case, Twitter was the resource and the City Cloud was the third party, but we didn't have a user. In fact, the City Cloud was also the user. This is a problem for applications like the City Cloud who crawl Twitter for data and have no direct user interface to the user. We couldn't show the user a login screen for Twitter, so we needed to implement a workaround for this problem, which did not only occur with Twitter but also with Facebook and other modern web services. Instead of letting a user log in we decided to log ourselves in without any user interface. We used an HTML crawler to GET and POST HTML pages back to Twitter and succeeded in authenticating the City Cloud. Other web services followed similarly.

When developing building blocks for demo purposes we came across a security feature of modern day browsers called "same origin policy". This means that webpage on www.a.com cannot call resources on www.b.com. This mechanism bears a particular significance for modern web applications that extensively depend on HTTP cookies to maintain authenticated user sessions, as servers act based on the HTTP cookie information to reveal sensitive information or take state-changing actions. This is not such a big problem when a building block is deployed because at that time it runs on the same domain, but it is during testing and for external applications. Allowing access from other domains is just what we want in this case. There are a couple of workarounds for this and one of the most well known is the use of a JSONP call back to inject script dynamically. In this way it is possible to call our Odata webservice from a different domain.

5.2 WEBSITE

The website had a fairly low priority, as it is just a portal displaying demos and making the power of the system apparent to other developers and end users. This meant the website was developed late, after most of the framework had been implemented. The website itself has been ported to the Umbraco CMS described in section 3.8 according to the specified design.

The web application, as discussed in section 4.5.1, has been built with the MVVM pattern in mind. The first version of this Silverlight application, however, was a rough draft not adhering to this design pattern but simply attempting to show the functionality. This proved to be a costly decision, since the graphical side of things required a major overhaul later on in the project. The OData Explorer[17] was a great example while building the web application since it showed how to parse the service and its metadata dynamically. This was a specific requirement for the application.

5.3 PHONE PLATFORM

The phone platform is a major part of the system. The combination of the building block server as part of the framework and a mobile application to provide access to those buildings blocks are key to the system's functionality. During the process of implementing all the features we originally planned on implementing we came across some problems.

To implement the mobile application in an efficient, clean and manageable way we already decided to stick with the MVVM design pattern during the initial design.

To be able to implement the mobile application in an efficient and clean way we decided to use the PRISM Library that has been developed by Microsoft.

First of all the original plan of combining HTML5 with our application as a host application to deliver a more native experience proved to be more challenging than we expected. We succeeded in making an HTML5 app communicating with a native windows phone host app. When investigating the way this could be done on other smartphone platform like Android and iOS we had to conclude that the differences are too big to make this work in a useable way. HTML5 is still very dynamic and there are some major differences in the way it is implemented across the platforms. Therefore we decided to put more focus on the portal experience of the mobile application and drop the integration with native experience part. For the applications that will require a native experience we

[17] http://www.odata.org/developers/odata-sdk

will also include links to native applications that are distributed through the marketplace from within the mobile application.

Another change to the mobile platform had to be made because of security considerations. We planned on providing hosting for dynamic building blocks through the cloud framework. The issue we encountered is that allowing third parties to submit executable code that runs from within your system cannot be achieved in a secure way. Implementing a system that analyzes every piece of submitted code to determine what it will try to do is simply unrealistic. Therefore we decided to only support HTML5 pages with javascript. Due to the nature of HTML5 those pages can still be very dynamic, but executing code on the server is impossible. To ensure developers still have full possibilities we changed the Building Block system to allow Building Blocks to be hosted on third party servers. A developer requiring dynamic pages with server side interaction can still decide to host his own application and deliver it through the city cloud.

6 TESTING AND QUALITY ASSURANCE

The City Cloud is a system designed for handling great amounts of data which can prove to be critical in certain situations. It is therefore detrimental to ensure a high quality of code and keep the number of faults to a minimum. This chapter will describe what methods are employed during the development of the City Cloud.

6.1 TEST METHODS

6.1.1 UNIT TESTS

A unit test is a way of testing an individual piece of a system. The programmer specifies input and the expected output for a specified method. Unit testing is a common method, but provides little insight if it does not achieve proper Code Coverage[18].

Where applicable, we developed unit tests per class and per method. These were written before the actual method was implemented, forcing a great understanding about the functionality of said method. No code was allowed to be checked in to the Team Foundation Server without passing the tests either, ensuring a functional build at all times during development.

6.1.2 CODE CONTRACTS

Code contracts is a .NET implementation of the Design by Contract (DbC) design pattern[19]. We used the DbC principle to control the state of the classes within the City Cloud framework. It is based on the concept of preconditions, postconditions and invariants to determine the state of a system. Code contracts can also be used to generate documentation. The use of code contracts has allowed us to find specific anomalies during development. DbC is a useful method when developing system which is characterized through extensibility, since every extension still has to adhere to the existing contracts.

[18] http://en.wikipedia.org/wiki/Code_coverage
[19] http://se.ethz.ch/~meyer/publications/computer/contract.pdf

6.2 SOFTWARE IMPROVEMENT GROUP ASSESSMENTS

6.2.1 FIRST ASSESSMENT

The following is the result of our first submission to the SIG.

[Aanbevelingen]
De code van het systeem scoort bijna 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De score wordt naar beneden gehaald door de Duplicatie, de Unit Size en de Unit Complexity.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden tussen de verschillende delen van de applicatie, maar ook binnen elk deel. Een voorbeeld van de eerste vorm van duplicatie is de AccountController die terug te vinden is in zowel het 'Framework' als het 'Web' deel. Binnen het Framework-deel is er bijvoorbeeld duplicatie tussen 'CityCloudMetadataProvider' en 'CloudCityMetadataProvider'. Het is aan te raden om dit soort duplicaten op te sporen en op te ruimen.

Een andere strategie is om te kijken naar bestanden met exact dezelfde naam (zowel in de normale code als de test code), dit komt ook enkele keren voor binnen het systeem. Het opruimen van dit type duplicatie zal er ook voor zorgen dat het totale volume van het systeem afneemt, wat ook weer bevorderlijk is voor de onderhoudbaarheid. Alhoewel dit systeem in absolute zin relatief klein is, is het vergeleken met andere bachelorprojecten een van de grootste projecten.

Het andere aspect waarop lager gescoord wordt is de Unit Size en de Unit Complexity, hierbij wordt gekeken naar het percentage code wat bovengemiddeld lang of bovengemiddeld complex is. Het systeem scoort lager dan gemiddeld op beide punten, wat onder andere veroorzaakt wordt door de extreem lange methodes in 'AddConnectorTransaction.cs' en 'LowProfileImageLoader.cs', waarbij de laatste ook nog eens bovengemiddeld complex is. Alhoewel er voor het laatste voorbeeld gezien de annotatie bewust voor is gekozen, is het toch aan te raden om deze (en andere lange/complexe) methodes in kleinere stukken op te splitsen. Dit zorgt ervoor dat elke methode makkelijker te begrijpen, te testen en daardoor te onderhouden wordt. Commentaar regels zoals '// Copy work items to private collections' en '// Process pending requests' geven meestal al aan dat er aparte blokken van functionaliteit te onderscheiden zijn.

Alhoewel het systeem nu nog redelijk scoort is het aan te raden om goed op bovenstaande punten te letten om ervoor te zorgen dat de onderhoudbaarheid niet gaat dalen. De aanwezigheid van test code lijkt er in ieder geval op te duiden dat het aanpassen van de normale code zonder al teveel moeite kan plaatsvinden.

A lot of these comments were still on our to-do list since the first assessment was in the middle of our development phase. Regardless, all points are valid and are addressed in the final version. The duplication issue arose after a rename of the project after which some files had not been correctly deleted from the Team Foundation server, so this issue was promptly corrected. The unit size and unit complexity has been toned down where applicable.

6.2.2 SECOND ASSESSMENT

The following is the result of our second submission to the SIG.

[Hermeting]
In de tweede upload zien we dat de omvang van het systeem is toegenomen en dat daarbij de score voor onderhoudbaarheid is verbeterd. De stijging van de score voor onderhoudbaarheid is toe te schrijven aan een flinke vermindering van de hoeveelheid duplicatie binnen het systeem. Daarnaast is er ook op het gebied van de lengte en complexiteit van de methodes vooruitgang te zien, er zijn geen extreem lange methodes meer te vinden en de complexiteit van de methodes is over het algemeen gedaald.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Het is goed om te zien dat naast een verbetering in de onderhoudbaarheid er ook nog steeds een stijging in het volume van de test-code te zien is.

We can conclude that we have improved the code in all aspects. There are no negative points left in this assessment, so our code is maintainable according to SIG standards.

7 ORGANISATION

Due to the high number of involved parties our project has an above average complexity from an organizational perspective. In this chapter we will describe in more detail the stakeholders and people involved, how we tried to keep communication effective and how we combined the available resources in an efficient dynamic planning. To make sure the project stayed on the right track throughout the process Tom was appointed as project manager. More on the other defined roles later in this chapter.

7.1 STAKEHOLDERS

7.1.1 Microsoft

The Imagine Cup is organized by Microsoft. At the worldwide finals we represent the Netherlands and therefore also in a way represent the Dutch Microsoft subsidiary. Maarten-Jan Vermeulen is Academic Relations Manager at Microsoft and together with Brian Hewitt provided us guidance throughout the project. In regular meetings both at the Microsoft office and the university we discussed project progress and priorities.

7.1.2 CapGemini

In the Netherlands CapGemini is sponsor of the Imagine Cup. They are supporting our project by providing access to the knowledge available within their company. In practice this means that in some of our meetings we would share and discuss our ideas with CapGemini employees that have professional knowledge on the subject matter. In case of technical challenges CapGemini could provide assistance. Marcel van der Burg was our primary contact.

7.1.3 Delft University of Technology

The university was also involved throughout the process. Both supporting us with the Imagine Cup itself and of course supervising the part of the project that would serve as a bachelor project. Under supervision of professor Arie van Deursen Felienne Hermans supervised the bachelor project and Tiago Espinha provided feedback and technical assistance. It’s worthwhile to note that both Felienne and Tiago have participated in the Imagine Cup themselves before.

7.2 HUMAN RESOURCES

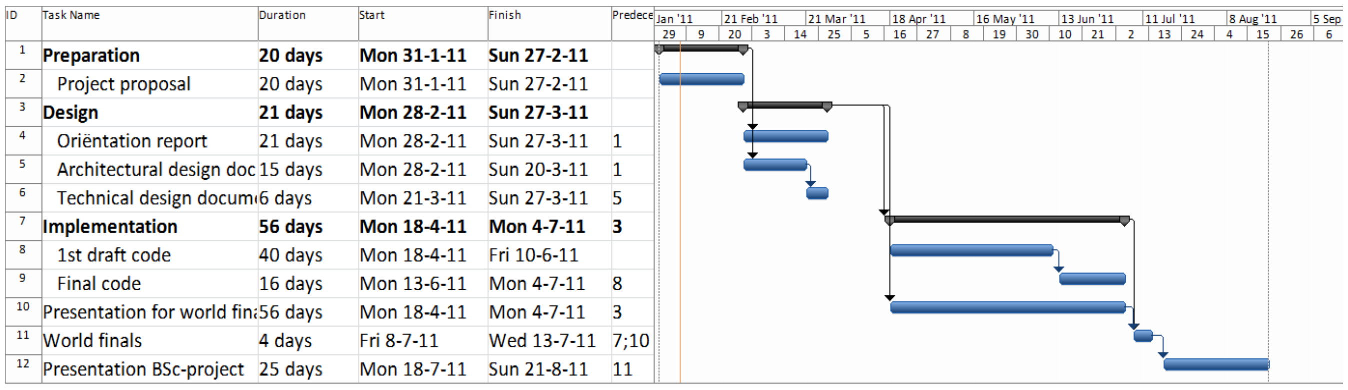
From day one the strength of our Imagine Cup team lied in the combination of students with different backgrounds. When bringing the team together we made sure all team members brought in skills that contribute to the overall skillset. Before moving on to the planning in the next section we will first introduce the other team members.

Jochem Toolenaar is a student in communication and multimedia design at the Rotterdam University of Applied Sciences. During our project he was working on his thesis about ubiquitous computing. Developing the City Cloud concept and being part of the team is one of the core parts of his thesis. Jochem is a valuable resource because he's good at developing a concept while still having enough technical knowledge to come up with feasible ideas. During the project he provided input whenever choices about functionality and design had to be made. Since his study also has a design background Jochem was also involved with designing the wireframes for the website.

Oana Nitu is a student currently pursuing a master degree in Business Information Systems at the Eindhoven University of Technology. She originates from Romania where she already achieved a bachelor degree in Computer Science together with a collection of Microsoft certificates. During her previous and current studies her focus has been on the management of IT systems which proved to be very useful. Oana provided us with useful feedback throughout the project. Her responsibility within the Imagine Cup team is the business case attached to our concept. Ensuring a viable business case played on important role throughout the implementation since effective use of resources in the cloud is required to keep the costs down.

7.3 PLANNING

When working on the project proposal back in February we defined an initial planning. It can be found in section 3.4 of Appendix A, but for convenience is also provided here.



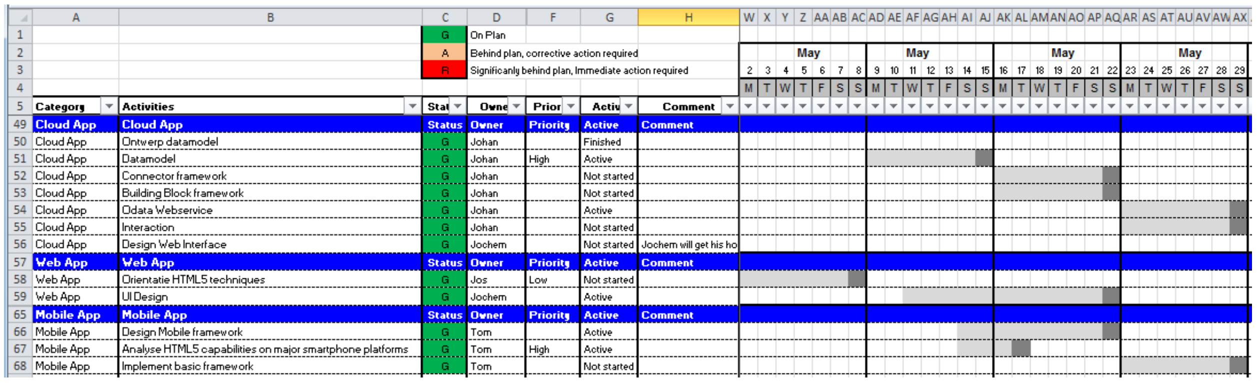
ID	Task Name	Duration	Start	Finish	Predece
1	**Preparation**	**20 days**	**Mon 31-1-11**	**Sun 27-2-11**	
2	Project proposal	20 days	Mon 31-1-11	Sun 27-2-11	
3	**Design**	**21 days**	**Mon 28-2-11**	**Sun 27-3-11**	
4	Oriëntation report	21 days	Mon 28-2-11	Sun 27-3-11	1
5	Architectural design doc	15 days	Mon 28-2-11	Sun 20-3-11	1
6	Technical design docum	6 days	Mon 21-3-11	Sun 27-3-11	5
7	**Implementation**	**56 days**	**Mon 18-4-11**	**Mon 4-7-11**	**3**
8	1st draft code	40 days	Mon 18-4-11	Fri 10-6-11	
9	Final code	16 days	Mon 13-6-11	Mon 4-7-11	8
10	Presentation for world fin	56 days	Mon 18-4-11	Mon 4-7-11	3
11	World finals	4 days	Fri 8-7-11	Wed 13-7-11	7;10
12	Presentation BSc-project	25 days	Mon 18-7-11	Sun 21-8-11	11

Figure 8. The initial project planning with deadlines.

This planning has been our guideline throughout the project. Since we used agile software development a more specific planning was not defined in advance. Once every few days we had a short meeting where we discussed what feature to implement next. It is key to agile development that new features are added to the system incrementally in short iterations. Especially in a project like ours where priorities change rapidly this proved to work out pretty good.

All three of us are used to agile development and the reasoning behind it. In practice this allowed us to shift focus to parts of the system that required extra attention when necessary. When working on the core framework during the first few weeks we used pair programming. The design of the core framework impacts all other parts of the system and therefore needs to be designed carefully. By using pair programming there is always two people working on a piece of code together. This will ensure better code quality and a well thought design.

Throughout the project we used a special spreadsheet to keep track of the different tasks and responsibilities. This sheet provided instant insight in the status of different tasks and allowed people to work without spending too much time communicating. We highly recommend using a tool like this, especially when working with a team that's spread around different locations.



G	On Plan
A	Behind plan, corrective action required
R	Significanly behind plan, Immediate action required

	Category	Activities	Stat	Owne	Prior	Activ	Comment
49	**Cloud App**	**Cloud App**	**Status**	**Owner**	**Priority**	**Active**	**Comment**
50	Cloud App	Ontwerp datamodel	G	Johan		Finished	
51	Cloud App	Datamodel	G	Johan	High	Active	
52	Cloud App	Connector framework	G	Johan		Not started	
53	Cloud App	Building Block framework	G	Johan		Not started	
54	Cloud App	Odata Webservice	G	Johan		Active	
55	Cloud App	Interaction	G	Johan		Not started	
56	Cloud App	Design Web Interface	G	Jochem		Not started	Jochem will get his ho
57	**Web App**	**Web App**	**Status**	**Owner**	**Priority**	**Active**	**Comment**
58	Web App	Orientatie HTML5 techniques	G	Jos	Low	Not started	
59	Web App	UI Design	G	Jochem		Active	
65	**Mobile App**	**Mobile App**	**Status**	**Owner**	**Priority**	**Active**	**Comment**
66	Mobile App	Design Mobile framework	G	Tom		Active	
67	Mobile App	Analyse HTML5 capabilities on major smartphone platforms	G	Tom	High	Active	
68	Mobile App	Implement basic framework	G	Tom		Not started	

Figure 9. The task spreadsheet.

7.4 COMMUNICATION

When dealing with different stakeholders and team-members geographically located somewhere else efficient communication becomes increasingly important. As a project manager it was Tom's role to make sure everybody got all the information they needed when it was needed. This holds both for stakeholders and team members. To achieve this goal different means of communication were used. Regular update e-mails where sent to everyone involved with the project.

Next to regular e-mail updates project progress was also discussed in a weekly call with Microsoft. During this call a few topics were always discussed: The current status, short-term planning for the current week and an update to the long-term planning when necessary. Once in a while the team had a meeting at Microsoft. During those meetings all produced material was reviewed and discussed. Meetings at the Microsoft office where usually also joined by our main CapGemini contact Marcel van der Burg.

Our daily supervisor at the Delft University, Felienne Hermans, also received extra updates next to the regular e-mail updates. Project progress was also communicated through the bi-weekly progress reports and a meeting once in a while.

Within our team we used Microsoft Lync an important addition to communication via e-mail. In a weekly online Lync call with the full team we discussed full project progress, important decisions and short- and long-term planning. Once every two or three weeks the Lync call was replaced by a real-life meeting.

8 CONCLUSIONS

We think we have done well, given the scale and complexity of the system. The notion of Open Data is gaining popularity by the day, which gets made more clear by statements made by Neelie Kroes (Kroes, 2010) and the city of New York (Judd, 2011). The system is up and running, although we did not achieve every feature we would like to have achieved, it runs well. We have partnered with the city of Rotterdam to demonstrate the system there, and their eagerness shows just another of potential the City Cloud has.

On the topic of process and project management we are extremely happy. We have been under slight stress nearing the deadline of implementation for the Imagine Cup finals, but other than that we have thoroughly enjoyed ourselves without meeting large obstacles along the way. It has been a long and worthwhile journey for us as bachelor students as well the other members involved in the Imagine Cup team.

9 RECOMMENDATIONS

As expected, the project was too large to finish within a couple of months. We have therefore not implemented some of the features we had designed. This section will describe the different features which still have to be finished before the City Cloud can be deployed in a real life setting.

First off, the authentication subsystem has to be completed. The design and groundwork for this is in place, but the Azure Access Control Services has to be combined with the City Cloud to provide a seamless login to the user. This should be possible through the mobile application as well as the website. We recommend using OAuth, so users can login with their Facebook account.

Secondly, the dynamic loading of connectors and building blocks should be finished. This is implemented entirely already, except for putting the new elements into the queue storage on Azure from the website. This is a simple matter of time which we unfortunately could not finish in time for the Imagine Cup finals.

Thirdly, most of the incoming and outgoing request the framework performs can be optimized by implementing batch requests. This could reduce a lot of traffic, and will become very significant if the system becomes larger. It is recommended to monitor connectors as well to see if they use batch requests properly.

To further improve traffic, a caching system would be a great improvement. This means the City Cloud stores the retrieved data temporarily so similar requests do not have to be sent multiple times as long as it is within a short timeframe.

Finally, as a 2.0 feature, it would be nice if end users could add data back into the system as well. This is currently not supported and will require a great expansion of the system before it could be possible. OData does support PUT and POST requests so there are no technical limitations preventing it from happening, however.

BIBLIOGRAPHY

Coley Consulting. (2011). *MoSCoW Prioritisation*. Retrieved 8 1, 2011, from Coley Consulting: http://www.coleyconsulting.co.uk/moscow.htm

Fowler, M. (2004). *Presentation Model*. Retrieved 8 1, 2011, from Martin Fowler: http://martinfowler.com/eaaDev/PresentationModel.html

Judd, N. (2011, 5 16). *New York Releases 'Road Map for the Digital City'*. Retrieved 8 1, 2011, from techPresident: http://techpresident.com/short-post/new-york-releases-road-map-digital-city

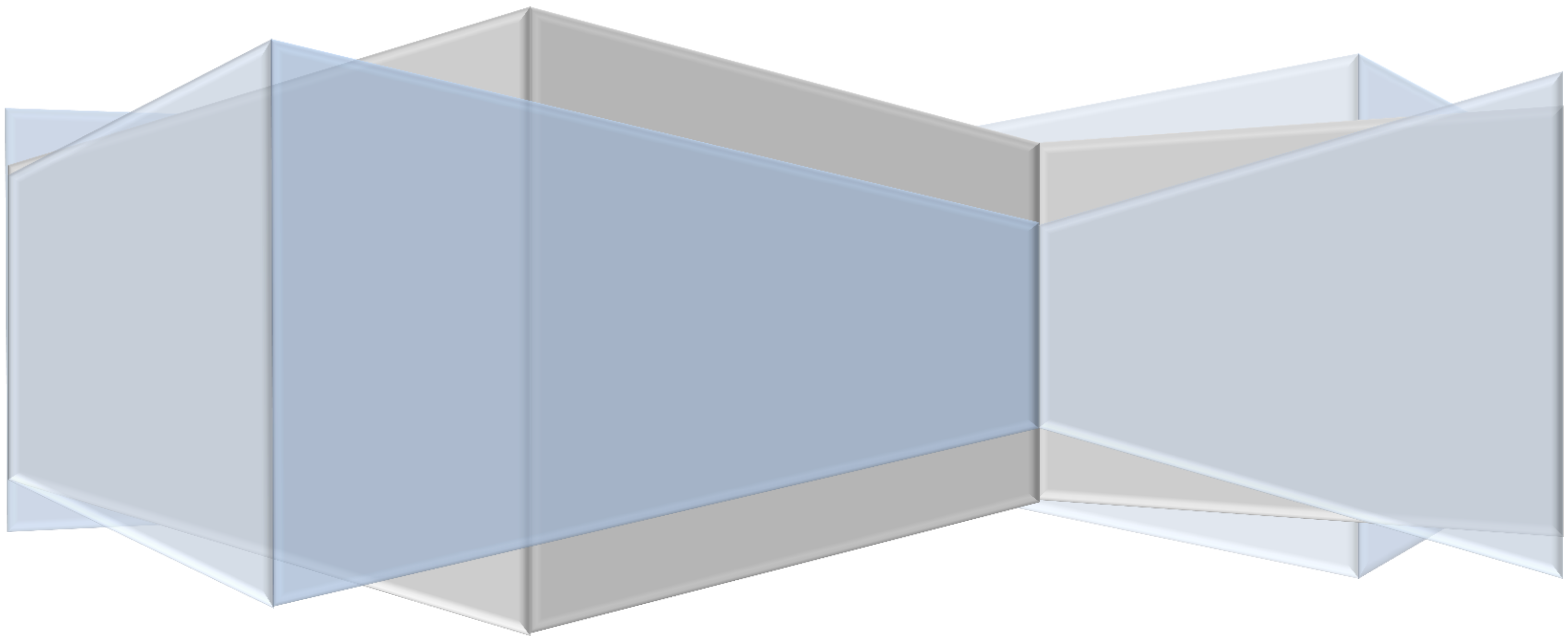
Kroes, N. (2010). My vision for eGovernment, and how to make it real. *"Lift-Off towards Open Government" conference.* Brussels.

Microsoft. (2010). *UI Design and Interaction Guide for Windows Phone 7*. Retrieved 04 20, 2011, from http://go.microsoft.com/fwlink/?LinkID=183218

APPENDICES

Bachelor Project Proposal

Jos Kraaijeveld
Johan Laanstra
Tom Verhoeff



APPENDIX A

CONTENTS

APPENDIX A

1 INTRODUCTION

1.1 MOTIVATION

In our BSc-project we want to combine working with a big company with actually building something useful. Microsoft provides just that with the Imagine Cup, a global student technology competition with over 300000 participants. Microsoft NL offered us the chance to work with them towards a solution that will be presented at the worldwide Imagine Cup finals in July. For us this is an opportunity to work together with people from both Microsoft and CapGemini on the full process of developing a software solution.

1.2 BACKGROUND

The theme for Microsoft's Imagine Cup is "Solving the world's toughest problems" referring to the UN Millenium Goals, so we tried to find a problem that could realistically be solved with a software solution developed within the specified amount of time for a BSc-project. With our project we want to support global development by providing a framework that supports a municipality to involve their citizens and companies in bringing the city's services to the cloud.

2 PROJECT DESCRIPTION

2.1 OBJECTIVE

The main objective of our BSc-project is to implement the software side of this year's Dutch entry in the software design category of Microsoft's Imagine Cup. This year's entry will be focused on creating a cloud-platform to support municipalities and their citizens in their daily business. Our objective is to support the conceptual phase and provide implementation of the base system. The final concept will be defined before implementation starts.

2.2 SCOPE

Based on the current concept the scope of our project will be the following parts:

1. Design and build the cloud-based framework
2. Design and build a prototype mobile and web application
3. Design and build an example building block

2 and 3 will be implemented to test and prove the functionality of 1. Spare time will be used to expand the implementation of 2 and 3, but this will not be a part of the intended project scope.

2.3 DELIVERABLES

- Preliminary Report
- Requirements analysis document
- Architectural design document
- Final Report
- Cloud, mobile and web application with source-code

2.4 CONSTRAINTS

- The Imagine Cup team also consists of one student from the Eindhoven University and one from Hogeschool Rotterdam. The final concept will be decided together with the rest of the team and is therefore subject to minor changes which might influence the final outcome of the project.

3 APPROACH

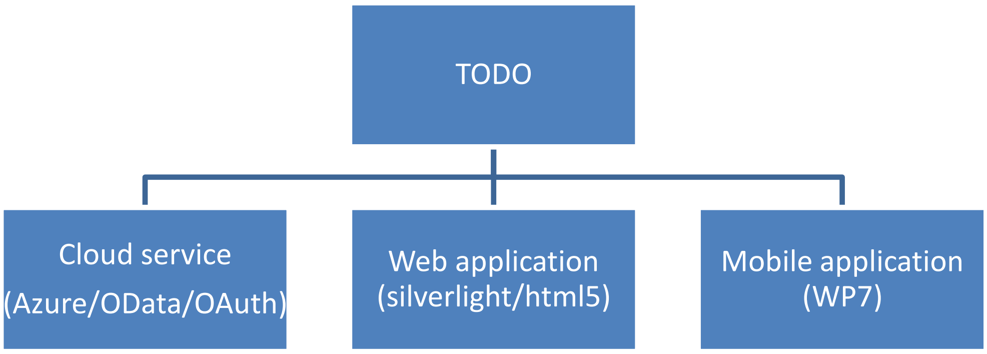
3.1 METHODOLOGY

Because not all group members and other people involved are located at the same place a Windows Live Group was started to collaborate and share ideas and documents. For the development of the software we will employ a form of agile software development. This allows for an iterative process and facilitates the steady development by adding new features and building blocks step by step.

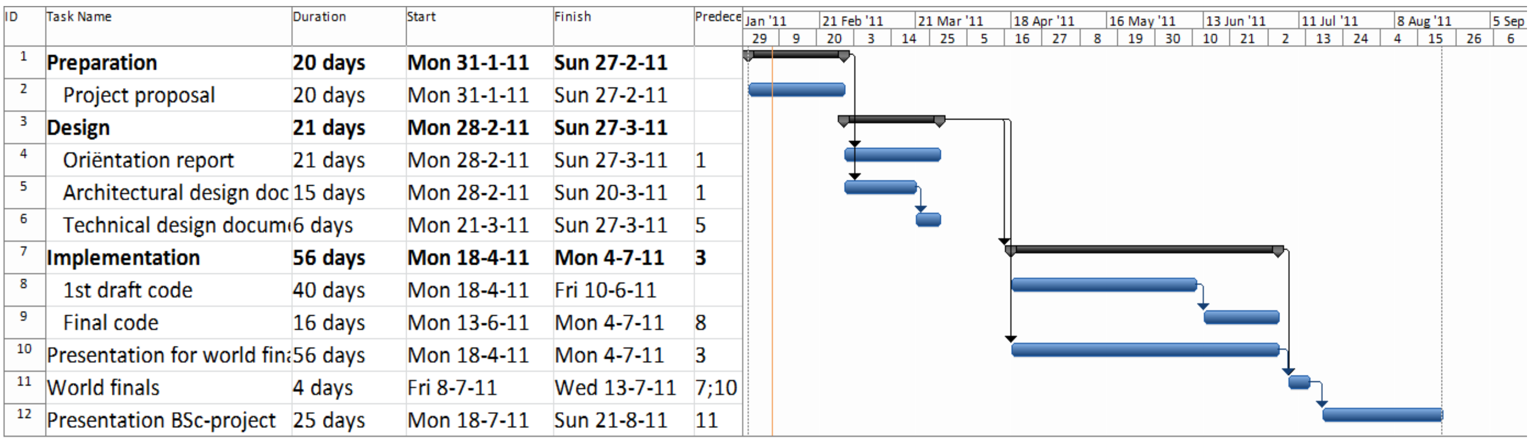
3.2 TECHNIQUES

The code will be written in the C# programming language combined with the .NET Framework. In this way the software can be developed for different platforms, such as Windows Phone 7, Windows Azure, Windows 7, Silverlight. The Visual Studio Integrated Development Environment is used for this which incorporates tools for codestyle checking, unit tests and other best practices. For communication between different parts of the system, language independent protocols will be used so it can be easy to port parts of the system to other platforms.

3.3 WORK BREAKDOWN STRUCTURE


TODO
Cloud service (Azure/OData/OAuth)
Web application (silverlight/html5)
Mobile application (WP7)

3.4 PLANNING



ID	Task Name	Duration	Start	Finish	Predece
1	**Preparation**	**20 days**	**Mon 31-1-11**	**Sun 27-2-11**	
2	Project proposal	20 days	Mon 31-1-11	Sun 27-2-11	
3	**Design**	**21 days**	**Mon 28-2-11**	**Sun 27-3-11**	
4	Oriëntation report	21 days	Mon 28-2-11	Sun 27-3-11	1
5	Architectural design doc	15 days	Mon 28-2-11	Sun 20-3-11	1
6	Technical design docum	6 days	Mon 21-3-11	Sun 27-3-11	5
7	**Implementation**	**56 days**	**Mon 18-4-11**	**Mon 4-7-11**	**3**
8	1st draft code	40 days	Mon 18-4-11	Fri 10-6-11	
9	Final code	16 days	Mon 13-6-11	Mon 4-7-11	8
10	Presentation for world fin	56 days	Mon 18-4-11	Mon 4-7-11	3
11	World finals	4 days	Fri 8-7-11	Wed 13-7-11	7;10
12	Presentation BSc-project	25 days	Mon 18-7-11	Sun 21-8-11	11

APPENDIX A

4 PROJECT ORGANIZATION

4.1 INVOLVED PARTIES

TU Delft – Microsoft – Capgemini

The imagine cup is organized by Microsoft and therefore Microsoft is involved in the project. They support the team and provide the tools and resources. CapGemini is a partner of Microsoft and will support the team during development.

4.2 HUMAN RESOURCES

The team exists of Johan Laanstra, Tom Verhoeff and Jos Kraaijeveld. One master student attending the Eindhoven University of Technology and a student of the Hogeschool Rotterdam will also be supporting the Imagine Cup team.

5 QUALITY CONTROL

5.1 QUALITY

Code checking and best practices tools will be used to maintain the quality of the code.

5.2 DOCUMENTATION

Documentation will be created where needed and where appropriate.

5.3 VERSION CONTROL

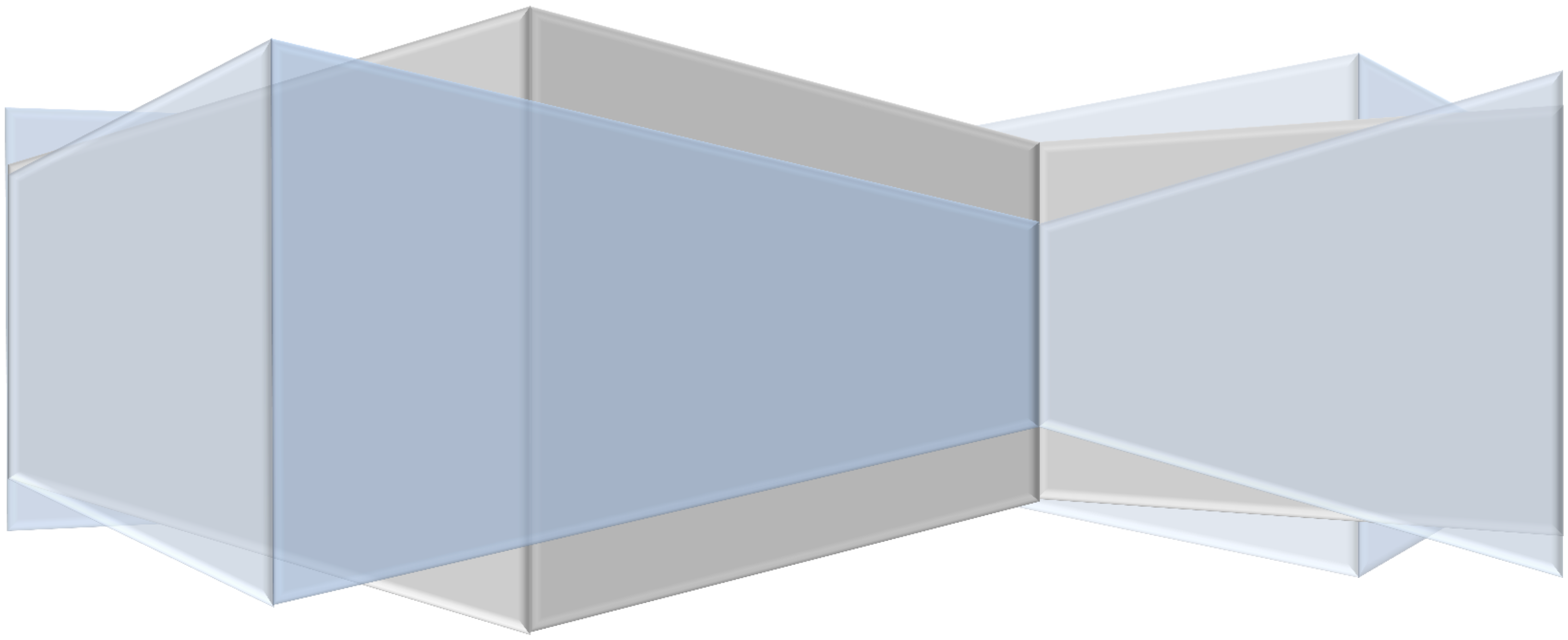
For version control we use Visual Studio Team Foundation Server 2010 for all source code and Windows Live Groups with Windows Live Mesh to share documents between group members.

5.4 MONITORING AND EVALUATION

There will be weekly meetings with the whole team to discuss progress and created the planning for the coming week.

O!ife Research Document

Jos Kraaijeveld, Tom Verhoeff en Johan Laanstra



APPENDIX B

CONTENTS

APPENDIX B

1 INTRODUCTION

A city is an evolving entity, an evolution kept in motion by the people who live in it. Our cities are in constant development; old buildings are taken down and new buildings take their place. But in the last decade or so our city has seen a different kind of development. As more and more information processing capabilities are embedded throughout the urban environment, the city is becoming smarter.

All this information is, however, left widely unused in current information societies. Most of the data a city generates is closed and scattered. As an example, social media generate significant amounts of data about the city, its inhabitants and day-to-day business, but the city does not efficiently use this data yet. The city does not play any role within the digital lives of its inhabitants.

O!ife is meant to improve the usage and re-usage of present and future information. Open data offers lots of possibilities for citizens, municipalities and companies within the city. By actively contributing and using data offered by O!ife, the city can evolve digitally and city development will accelerate.

In the future, even more information will be generated by a city, like the air pollution sensors (MIT Senseable City Lab, 2009) and the 'smart doors'. There are many more sensors and embedded software which will be part of the city soon. More examples can be found in the book Sentient City by Mark Shepard (Shepard, 2011).

The concept of O!ife fits Microsoft's Imagine Cup theme, "Solving the world's toughest problems", very well, especially the following Millennium Goal:

"8f. In cooperation with the private sector, make available benefits of new technologies, especially information and communications". (United Nations, 2011)

2 MOTIVATION

O!ife is all about open data. It is about a framework in the city that facilitates contributing and using of all kinds of data. In this document we are going to do some research in techniques and tools that can be used to make data available and open.

Chapter 4 will be about information sources. Which sources can we use and what data is currently available? Chapter 5 will be about research into the system that is responsible for data storage. Such a system should be very flexible and extendable. Chapter 6 will look into further possibilities of Windows Azure and chapter 7 will describe two demo applications to show the data. Finally, we look at similar projects in chapter 8.

APPENDIX B

3 GLOSSARY

- BLOB – Binary Large Object.
- Building block – An HTML widget which describes an application running within the O!ife cloud.
- Closed data – Data which is not publically available.
- Cloud computing - The provision of computational resources on demand via a computer network, just like an electricity network provides electricity.
- Connector – A software package which connects a third party data source to the O!ife cloud.
- Geodata – Data describing a certain location on Earth.
- Open data – Data which is publically available.
- Ubiquitous computing – The notion that information processing is embedded into everyday objects.

4 SOURCES OF INFORMATION

Our concept especially relies on information of which there is a lot available today. Careful analysis is required to get a good overview of the possible sources of information. In this section, we look at different sources and what information we can obtain from these sources. We also look at ways to obtain this information, and ensure no privacy or security boundaries are crossed.

4.1 SOCIAL MEDIA

Social media are one of the biggest available and public information sources today. In this section, we analyze which forms of social media are applicable for O!ife. We determine what information can be accessed and what means are required to retrieve it.

4.1.1 FACEBOOK

Facebook is the biggest social networking website in the world. There are currently more than 500 million active users which share over 30 billion pieces of content each month. (Facebook, 2011)

API: Yes
Authentication: OAuth
Request Method(s): HTTP GET/POST
Output Data Format(s): JSON
Available data: All public Posts / People / Pages / Events / Groups / Places / Check-ins.

Of the available data, we use Events, Places and Check-ins as the main source of data from Facebook.

4.1.2 TWITTER AND TWITPIC

Twitter is a quickly growing social broadcasting service. Users can broadcast small messages to everyone. Its users generate 140 million so-called *tweets* every day. Together with geodata a tweet can provide useful insight regarding citizens. (Twitter, 2010)

TwitPic is the image repository linked to Twitter. Pictures can be shared in real-time on Twitter using this service. (TwitPic, 2011)

API: Yes
Authentication: OAuth
Request Method(s): HTTP GET/POST
Output Data Format(s): JSON / RSS / Atom
Available data: Tweets / Retweets / Trends / Local trends / Direct messages / Geodata / Images / Videos/ User information / Comments / Media at specified location / Images with a specified user tagged / Events created by user / Images with tag
Extra notes: Rate limiting applies, meaning high load can prove to be an issue.

Local trends and tweets combined with geodata provide the information we require from Twitter. From TwitPic we use images which are geotagged to help create a digital overview of the city.

4.1.3 FOURSQUARE

Foursquare is a location-based mobile platform that makes cities easier to use and more interesting to explore. By “checking” via a smartphone app or SMS, users share their location with friends while collecting points and virtual badges. It has over eight million users worldwide and over 2.5 million check-ins per day. (Foursquare, 2011)

API: Yes
Authentication: OAuth
Request Method(s): HTTP GET/POST
Output Data Format(s): JSON / JSONP
Available data: Information / photos of venues, check-in information, tips.
Extra notes: Rate limiting applies, meaning high load can prove to be an issue.

We can use Foursquare to monitor crowdedness in specific areas. A crowded area can indicate there is an event happening there. The photos and information of venues will be used to support the online map of the city.

4.1.4 FLICKR

Flickr is an online photo management and sharing web application. It helps users make photos available to people who matter to them. Its users upload over five thousand images each minute. Currently, there are over 140 million geotagged images. (Flickr, 2011)

API: Yes
Authentication: OAuth
Request Method(s): REST / XML-RPC / SOAP
Output Data Format(s): REST / XML-RPC / SOAP / JSON / PHP
Available data: User post activity / Blogs / Collections / Contacts / Favorites / Galleries
Groups / Interestingness / Photos based on Tags / Geodata / Contexts / Info / Dates

From Flickr, we use the data from interestingness, photos based on tags and geodata. They can contribute to a digital overview of the city.

4.2 PUBLIC SECTOR

A common thought is that the governments have a lot of unused data stored within their systems. As it is not clear how much data it is, we look at the different types of initiatives and sources of information regarding public sector data. Some governments already provide citizens with full insight into public documents, but most of them are reluctant to make them open.

The *Open Government Data Initiative* (OGDI) (Microsoft, 2009) is a relatively new initiative which encourages governments, districts, cities and more to publish their documents available for everyone as well. Unfortunately, this has not been adopted widely yet. It is, however, a proof of concept: a lot of data, which can be relevant for various uses, can be available without crossing privacy boundaries. Great examples can be found on the European Open Public Data website (Microsoft, 2011).

The OGDI has information on a national level. A lot of data generated on a lower level, like crime rates in certain areas, are closed or only released once a year. Information on a lower level, per city or town, can allow for more uses. Some of the city data is released, but this differs per city and all the information gets published in different places.

Our goal is to organize the public sector data while staying conform to data publication guidelines, as described in 'Public Sector Data Sharing: Guidance on the Law'.

4.3 PRIVATE SECTOR

Using the private sector for the cloud is less of a priority. If the system is running, companies can freely contribute and obtain data from the cloud to help it grow. The types of data and ways to obtain it regarding the private sector are therefore irrelevant once the system is running.

APPENDIX B

4.4 SENSORS IN URBAN AREAS

Nowadays, more and more cities have small, embedded, technical applications which generate all sorts of data. Providing this data in the cloud allows developers to use this real-time information in their applications. An example of one of these sensor applications is the *smart door*, a door which counts how many people are passing by. This can be used for a crowdedness monitor, for instance. Another example is the use of bicycle tires which monitor air pollution (MIT Senseable City Lab, 2009). The air pollution information, combined with a specific location via GPS, can see all kinds of applications, like an application which informs the government of fires or unsafe areas in real-time.

4.5 INDIVIDUALS

Citizens produce all kinds of data, like short messages, photos and videos on social networks. Besides this, individuals can be a great source of information. Existing wearable body monitors and health trackers (Hanlon) can be used to monitor the overall health of the city. Smartphone GPS location can be used to monitor crowdedness. These are just two of many examples in which individuals can contribute to the overall cloud.

5 TECHNOLOGIES AND CHALLENGES

This section outlines the techniques that will be used for the system that handles data. The growing amount of data gives some challenges which we need to take into account.

5.1 CLOUD

The system needs to handle data, and it is likely that the amount of data the system will handle will grow significantly (Gantz & Reinsel, 2010). Cloud services are a good choice to take care of this problem. Cloud services like Windows Azure are a highly effective way to store large amounts of data like this project requires. It does not require a large investment in server infrastructure and is highly scalable and therefore cost efficient. A high uptime is guaranteed as well using a cloud service.

We have chosen for Windows Azure (Microsoft, 2011) for our data handling system for a couple of reasons. First of all, the Imagine Cup is a Microsoft contest, which promotes the use of Microsoft software. Besides that, we find that the development tools for Windows Azure are way better than the tools of the competitors and they offer great possibilities to build high-quality web applications in the cloud using the .net framework and C#. Unfortunately, it lacks various options like a mail service. We do not think it is necessary to have support for mailing in our web service, so we do not mind this disadvantage.

Other options include Amazon's EC2 Cloud (Amazon, 2011) and Google's App Engine (Google, 2011), but we have decided not to use these two. The EC2 Cloud offers a virtual server hosted at Amazon, which forces you to do a lot of administrative tasks. The App Engine is too restricted for our use since it forces you to use existing frameworks for a cloud application. The latter does not have a relational data store either, which we probably want to use. We did not make a choice for relation storage yet, but we want to keep the option open.

5.1.1 CHALLENGES IN THE CLOUD

Using cloud services offers a few other challenges. You do not have the usual local storage in Azure, where you can place assemblies and load them dynamically from your compilation folder. The problem is that the new assemblies are not part of the cloud application and are therefore not moved when the application is moved. We need this ability for our system to be extensible. To add extra data sources it should not be needed to redeploy the whole application, because that would take a lot of time, and moreover the system would not be usable during deployment. Azure offers an additional way to store binary large objects called blob storage. Research discovered that according to a Microsoft blog (Cohen-Yashar, 2011) it is indeed possible to load assemblies from blob storage.

5.2 ODATA

Web services are currently booming. Every big website with user generated content has its own web service, to allow developers to build applications for it. As our platform is mainly a developer platform to open up data, a web service is an important part of the data system. As with most technologies there is not a single way to do web services. Web services use all kinds of protocols to expose their data like SOAP, XML, REST, RSS, Atom, JSON and OData.

The Open Data Protocol (OData) (Microsoft, 2011) is a Web protocol for querying and updating data. OData does this by applying and building upon Web technologies such as HTTP, Atom Publishing Protocol (AtomPub) and JSON to provide access to information from a variety of applications, services, and stores. The protocol emerged from experiences implementing AtomPub clients and servers in a variety of products over the past several years. OData is being used to expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems and traditional websites.

The great advantage of using OData for our web service is the way you can query for data. You can filter in the URI and apply all kinds of relations in the query. WCF Data Services currently offer full support for the OData protocol. WCF Data Services is part of the .NET 4 Framework.

6 WINDOWS AZURE

Our application is mainly about the Cloud Computing aspect. For this reason, we include further research into Windows Azure.

Azure has three types of storage, namely Blob storage, Table storage and Queue storage. Each of them has different uses. In this section we show how we use the different types of storages for our application. All of these services can be accessed with a Windows Azure account, for which we receive a 256-bit secret authentication key. The Azure SDK also provides an easy way to access the different storages from a .NET program in the form of a library.

6.1 BLOB STORAGE

The Windows Azure Blob storage (Calder, Wang, Mainali, & Wu, 2010) can be seen as the section of a harbor storing containers. It is supposed to store large objects (up to 50 GB) in the cloud. It scales automatically in storage and load handling. A file, once in Blob storage, is called a blob. Multiple blobs can be placed inside a container to keep files organized. Sharing policies are set at the container level. Metadata is stored per container as well, in the <name, value> form with a maximum of 8KB per container.

APPENDIX B

Access to the Blob storage is done through a set of standard HTTP/REST commands. The set of commands is: {PUT Blob, GET Blob, DELETE Blob, Copy Blob, Get Committed Block List, Get Uncommitted Block List}. This is done on the following URL: *http://<account>.**blob**.core.windows.net/<container>/<blobname>*.

Objects larger than 64MB cannot be PUT into the cloud in one request. The block interface must be implemented to support larger files. Each block will get a unique ID in the blob scope and can be queried for individually.

6.2 TABLE STORAGE

Table storage is Azure's implementation of structured storage (Haridas, Nilakantan, & Calder, 2009). It supports enormous scalable tables in the cloud. The cloud automatically scales to thousands of servers once traffic increases. A table is bound to an Azure account and has the following properties.

A table has entities, the basic data entries, as rows and properties as columns. There are a few special properties. PartitionKey is the first property of every table which is used to distribute the table's rows across servers. The RowKey is the second property, which is the unique ID within a partition describing an entry. Both keys combined identify an item in a table. Every entity has a timestamp as well.

The Table storage in Azure is no-SQL. That is, the tables have no fixed schema. This implies that two entities can have different properties, and properties with the same name can het different value types. The supported types for properties are Binary, Bool, DateTime, Double, GUID, Int, Int64 and String. An entity cannot exceed 1 MB in size.

6.3 QUEUE STORAGE

The queue storage in Azure is a reliable message delivery system through which asynchronous updates can be dispatched. (Microsoft, 2008) It can be used to connect different components within the cloud and ensures that a message gets processed.

Messages can be put into a queue before the backend server will process them. A message has an ID, a visibility timeout, a PopReceipt and a Time-to-live.

Operations on the queue are done in HTTP/Rest with the following set of commands. {CREATE Queue, DELETE Queue, SET Queue Metadata} on http://<account>.**queue**.core.windows.net/<QueueName>. For messages, the commands and URL are {PutMessage, GetMessage, DeleteMessage, PeekMessage and ClearQueue} at http://<account>.**queue**.core.windows.net/<QueueName>/messages.

7 DEMO APPLICATIONS

To show the potential of the system we will build an application for smartphones and a web application to show off the different data sets and to provide developers with guidance.

7.1 SMARTPHONES

The number of smartphones is increasing worldwide. More and more people carry on with them. Therefore smartphones are a good platform to target. Windows Phone currently has a market share of about 8% in the US. Therefore we also need to consider other mobile platforms like Android and iPhone. This is a bit of a challenge since we need developers to build building blocks that work on all mobile platforms.

A solution came up when Microsoft announced HTML5 support on Windows Phone 7 in the coming Mango update at MIX11. IPhone and Android already support HTML5 so this offers an opportunity. Loading HTML5 applications from the cloud in a browser control is supported on all major smartphone platforms. This potentially raises problems that there is no integration possible between the app and the building blocks.

Research on this topic resulted in the fact that there is two-way communication possible between the browser control and the app that hosts it on all major platforms (Google, 2011) (MSDN, 2011) (Apple, 2011).

7.2 WEB APPLICATION

It is difficult for a developer to create a good application without knowing what data can be found and used. The web application will serve as a visualization of the available data, displaying a map with data points plotted onto it. This can be filtered and ordered according to the user's wishes, and allows for quick access to the huge amount of data. All the data types will be explained here.

The web application will also serve as our distribution platform. Developers can register to get access and download tools we will offer. Submitting their applications or data feed connectors can also be submitted on the web application. Lastly, there will be a forum for users and developers and guides on how to use the system.

It is possible to host web applications on the Azure server we would already use, so to make our lives easier we shall use this for the web application as well. The site itself will be made in HTML5.

8 COMPETING AND SIMILAR SOFTWARE

8.1 EVRYTHNG

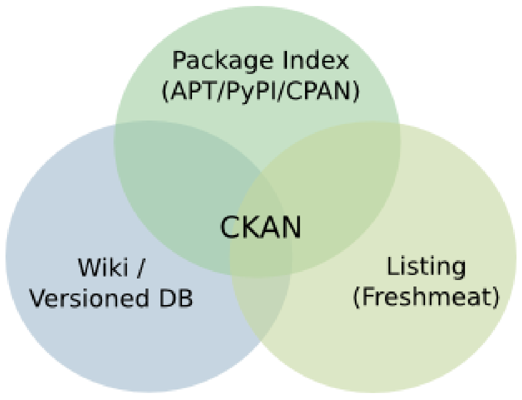
Evrythng (Evrythng, 2010) is a system in which every physical object can have its own online profile. They aim to connect every object to the internet as a unique online identity. It is a similar system, since they provide developers with the freedom to use the provided data for their applications. Our system specifically focuses on the city and its inhabitants. We also focus more on data instead of physical objects, being less limited regarding the input data.

8.2 CKAN

CKAN is the Comprehensive Knowledge Archive Network, a registry of open knowledge packages and projects (and a few closed ones), developed by the Open Knowledge Foundation (Open Knowledge Foundation, 2011). CKAN makes it easy to find, share and reuse open content and data**,** especially in ways that are machine automatable.

APPENDIX B

As a system CKAN functions as a synthesis of several different services:



First, thanks to its underlying versioned domain model CKAN has a wiki-like interface that lets anyone add and material held in it. Second, and unlike a wiki, CKAN can store 'structured' information, which allows it to provide 'index'-like features such as automated registration, discovery and installation of material. In this respect it behaves like CPAN or PyPI in the software world -- though again for open data and content not code.

CKAN differs from our proposed system in such a way that we provide a structured and easily reusable collection of data. CKAN simply stores data from different sources and does not alter its contents, which limits its usability.

8.3 AZURE DATAMARKET

DataMarket is a service that provides a single consistent marketplace and delivery channel for high quality information as cloud services (Microsoft, 2011). Content partners who collect data can publish it on DataMarket to increase its discoverability and achieve global reach with high availability. Data from databases, image files, reports and real-time feeds is provided in a consistent manner through internet standards. Users can easily discover, explore, subscribe and consume data from both trusted public domains and from premium commercial providers.

End users who need data for business analysis and decision making can conveniently consume it directly in Microsoft Office applications such as Microsoft Excel and Microsoft BI tools (PowerPivot, SQL Server Reporting Services). Users can gain new insights into business performance and processes by bringing together disparate datasets in innovative ways.

Application developers can use data feeds to create content rich solutions that provide up-to-date relevant information in the right context for end users. Developers can use built-in support for consumption of data feeds from DataMarket within Visual Studio or from any Web development tool that supports HTTP.

The DataMarket offers raw data for developers to use. We propose a system with a more concrete framework to allow easier development of applications. We also provide support for existing databases, which DataMarket does not.

APPENDIX B

9 BIBLIOGRAPHY

Amazon. (2011). *Amazon Elastic Compute Cloud*. Retrieved May 2, 2011, from Amazon Web: http://aws.amazon.com/ec2/

Apple. (2011). *UIWebView Class Reference*. Retrieved May 2, 2011, from iOS Developer Library: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIWebView_Class/Reference/Reference.html

Calder, B., Wang, T., Mainali, S., & Wu, J. (2010). *Windows Azure Blob.* Retrieved May 2, 2011, from Windows Azure Storage: http://www.microsoft.com/windowsazure/storage/

Cohen-Yashar, M. (2011, 01 24). *How To Load Assemblies From Blob*. Retrieved May 2, 2011, from Microsoft Blogs: http://blogs.microsoft.co.il/blogs/applisec/archive/2011/01/24/how-to-load-assemblies-from-blob-storage.aspx

Evrythng. (2010). *Home*. Retrieved May 2, 2010, from Evrythng: http://evrythng.net/index.html

Facebook. (2011). *Home*. Retrieved May 2, 2011, from Facebook Developers: http://developers.facebook.com/

Flickr. (2011). *The App Garden*. Retrieved May 2, 2011, from Flickr Services: http://www.flickr.com/services/api/

Foursquare. (2011). *Foursquare APIv2*. Retrieved May 2, 2011, from Foursquare Developers: https://developer.foursquare.com/

Gantz, J., & Reinsel, D. (2010). *The Digital Universe Decade - Are You Ready?* Framingham: EMC Corporation.

Google. (2011). *Run your apps on Google's infrastructure*. Retrieved May 2, 2011, from Google App Engine: http://code.google.com/appengine/

Google. (2011). *Using WebViews*. Retrieved May 2, 2011, from Android Developers: http://developer.android.com/resources/articles/using-webviews.html

Hanlon, M. (n.d.). *New wearable body monitor continuously measures calorific output*. Retrieved May 2, 2011, from Gizmag: http://www.gizmag.com/go/4106/

Haridas, J., Nilakantan, N., & Calder, B. (2009). *Windows Azure Table.* Retrieved May 2, 2011, from Windows Azure Storage: http://www.microsoft.com/windowsazure/storage/

Microsoft. (2008). *Windows Azure Queue.* Retrieved May 2, 2011, from Windows Azure Storage: http://www.microsoft.com/windowsazure/storage/

Microsoft. (2009). *Default*. Retrieved May 2, 2011, from Open Government Data Initiative: http://www.microsoft.com/industry/government/opengovdata/default.aspx

Microsoft. (2011). *Default*. Retrieved May 2, 2011, from European Open Government Data Initiative: http://www.govdata.eu/en/index.aspx

Microsoft. (2011). *Home*. Retrieved May 2, 2011, from Open Data Protocol: http://www.odata.org/developers/protocols

Microsoft. (2011). *Home*. Retrieved May 2, 2011, from Windows Azure Marketplace DataMarket: https://datamarket.azure.com/

APPENDIX B

Microsoft. (2011). *What is Windows Azure?* Retrieved May 2, 2011, from Windows Azure: http://www.microsoft.com/windowsazure/

MIT Senseable City Lab. (2009, 12). *About*. Retrieved May 2, 2011, from Copenhagen Wheel Project: http://senseable.mit.edu/copenhagenwheel/index.html

MSDN. (2011). *WebBrowser.ScriptNotify Event*. Retrieved May 2, 2011, from Silverlight API: http://msdn.microsoft.com/en-us/library/system.windows.controls.webbrowser.scriptnotify(v=vs.95).aspx

Open Knowledge Foundation. (2011). *About*. Retrieved May 2, 2011, from CKAN: http://ckan.net/about

Shepard, M. (2011). *Sentient City.* Boston: MIT Press Ltd.

TwitPic. (2011). *API Documentation*. Retrieved May 2, 2011, from TwitPic API: http://twitpic.com/api.do

Twitter. (2010). *FrontPage*. Retrieved May 2, 2011, from Twitter API Wiki: http://apiwiki.twitter.com/w/page/22554648/FrontPage

United Nations. (2011). *Develop A Global Partnership for Development*. Retrieved May 2, 2011, from United Nations Millenium Development Goals: http://www.un.org/millenniumgoals/global.shtml

APPENDIX C

City cloud

System Design Document

Johan Laanstra
Jos Kraaijeveld
Tom Verhoeff

APPENDIX C

CONTENTS

APPENDIX C

INTRODUCTION

This system design document is Olife's attempt to outline requirements, basic functionality and underlying data model for the application City Cloud. It is an initial design document which specifies the minimum required before we start an agile development process. The information in this document is prone to change during the project.

In the Actor identification section, we describe the different parties involved when the City Cloud is operational. We base our functional and non-functional requirements on all these types of users. Based on the requirements, we describe a minimal amount of use cases to cover the basic functions of the system. Afterwards, we propose a subsystem decomposition and the first concept for a data model concerning the Cloud system. Lastly, we describe our test and implementation plan.

ACTOR IDENTIFICATION

END USERS

Like any system, the City Cloud should cater end users with functionality which makes their lives easier. We define end users as users who use applications which in turn use our data cloud. They are inhabitants of a town who would like to have more services available to them in an easy way.

DEVELOPERS

To provide the end users with new services, we need the help of other software developers. They are the ones that will actively use the data made public in clever ways to write new applications. We need to ensure they have everything they need to do their work.

PROVIDERS

The providers are on the other end of the spectrum: they are the ones giving information. Every provider helps the system grow and gives developers more options for their applications. This group needs a low entry barrier so people share data more easily.

MAINTAINERS

The City Cloud developers, Olife, are the maintainers of the system.

REQUIREMENTS

The City Cloud system knows many actors. This section describes what functionality the system should offer to each of the actors described in the previous section, as well as further non-functional requirements that provide more constraints.

FUNCTIONAL REQUIREMENTS

END USERS

1. End users must be able to visualize the available data.

2. End users must have access to applications which use the public data on their mobile devices, depending on their location.
3. End users must be in control of their own data regarding the system.
4. End users must be able to log in.
5. End users should be able to see what new applications have been created since their last use of the system.
6. End users should be able to notify the maintainers when a certain application contains profanity.
7. End users could rate applications and publish them on social media.

DEVELOPERS

8. Developers must be able to have a clear view of available data types within the system.
9. Developers must be able to query and filter data within the system.
10. Developers must be able to use the public data in external applications.
11. Developers must be able to submit applications which become available on end users' mobile devices.
12. Developers must be able to submit data to the system from their applications.
13. Developers should be able to receive statistics about applications they have submitted.

PROVIDERS

14. Providers must be able to connect their current data storage systems to our system.
15. Providers must be able to submit data to the system.
16. Providers must be able to control their own data (e.g. remove/modify data which belongs to them).
17. Providers should be able to receive statistics about the usage of their data.

MAINTAINERS

18. Maintainers must be able to approve and deny submitted applications.
19. Maintainers must be able to remove approved applications.
20. Maintainers should be able to receive statistics about the entire system.

NON-FUNCTIONAL REQUIREMENTS

1. The system must return give a response within 10 seconds in 99% percent of all queries.
2. The visualization of data must work in every modern browser.

USE CASES

The City Cloud system has different actors as described before. These actors use the system in different ways, so we provide use cases for each actor. This section describes how every actor will use the system to achieve goals, giving insight into user-system interaction. Any additional use cases will be decided upon during the development process, according the Agile Development teachings.

END USERS

APPENDIX C

LOGGING INTO THE SYSTEM THROUGH A WEB PAGE

Goals: Getting access to the system.
Precondition: The person which tries to log in is required to have an account known to the system. The user must use a supported browser.
Summary: The user has to log in before data can be seen, used or submitted. A simple procedure is required for this. Afterwards, the user will be logged in and the system will be accessible.
Steps:

1. The user navigates to http://city-cloud.eu.
2. The user clicks 'Login with Facebook' or 'Login'.
3. The system returns the login page.
4. The user puts in his login credentials and presses 'Log in'.
5. The system returns if the login was successful. If it was, it returns the data visualization home page. If it wasn't, the user is prompted with the same login page.

LOGGING INTO THE SYSTEM VIA A MOBILE DEVICE

Goals: Getting access to the system.
Precondition: The user has a mobile device with a supported operating system. The user has the custom City-Cloud application installed.
Summary: The user needs to log in before he can use location-based applications created by developers. The mobile application will prompt the user for login. Afterwards, the user will be logged in and will be presented available applications.
Steps:

1. The user starts the mobile City Cloud application.
2. The system requests login, through Facebook or regularly.
3. The user puts in login credentials and presses 'Log in'.
4. The system returns if the login was successful. If it was, the application home screen is shown with highlighted new available applications. If it wasn't, the user is prompted for credentials again.

VISUALIZE DATA

Goals: Getting insight into the available data and plotting these onto a map.
Precondition: The user must be logged in and must use a supported browser.
Summary: The user uses a web interface to select certain data sets to display it onto a specific region on a map.
Steps:

1. The user navigates to http://city-cloud.eu.
2. The user clicks on 'Explore the data'.
3. The system returns the data set explorer and map.
4. The user navigates to a certain region on the map and selects the wanted data sets.
5. The system displays tables with data and plots the location-based data on the map, available for a mouse over action.

RUN A MOBILE APPLICATION

Goals: Running the mobile wrapper application which displays all the available developer-made applications in a region.
Precondition: The user must run the application on his mobile phone and must be logged in.
Summary: The user uses his mobile phone to navigate to the mobile application and browses the available applications within his region.
Steps:

1. The user selects the City Cloud application on his mobile phone.
2. The system returns the available applications.
3. The user selects an applications.
4. The system loads the developer-made application and presents this to the user.

DEVELOPERS

USE THE DATA IN AN APPLICATION

Goals: Using the City Cloud to feed information into an application.
Preconditions: The developer has an access token.
Summary: The developer uses the provided API and libraries for different programming languages to query the City Cloud.
Steps:

1. The developer writes his application and calls the library function as specified by the online documentation.
2. The City Cloud returns dataset objects according to the data model specified by the online documentation.
3. The developer reads the data and display it accordingly in his application.

SUBMIT AN APPLICATION

Goals: Get an application visible and usable from within the City Cloud Mobile Application.
Preconditions: The developer has an access token.
Summary: The developer attempts to submit his application to the system, after which it gets checked by the maintainers and published if it does not breach any regulations.
Steps:

1. The user goes to http://city-cloud.eu and presses ‘Submit application’.
2. The system displays the steps required to upload an application.
3. The user uploads his application to the system.
4. The maintainers check the application to ensure regulations are enforced.
5. If the application passes the check, the developer gets informed and the application becomes visible for users starting the mobile application. If it doesn’t, the developer gets notified with a reason of denial.

SUBMIT DATA TO THE SYSTEM THROUGH AN APPLICATION

Goals: Submit new data to the system through an application.
Preconditions: The developer has an access token.
Summary: The developer uses the libraries or API calls to submit information received through an application

back into the system.
Steps:

1. The developer develops an application and determines new types of data based on what users do with his application.
2. The developer uses the given libraries or API calls to submit data into the City Cloud from the mobile application.
3. The system displays a confirmation box the first time a user uses the application, stating which information will be sent when using the application.

PROVIDERS

CONNECT AN EXISTING INFORMATION SOURCE TO THE SYSTEM

Goals: Make specified information publically available through the City Cloud.
Preconditions: The person in charge of the connecting process understands the source and the connector interface.
Summary: An existing source of data will be made queryable through the City Cloud through a connector, which specifies the data types in the source and translates the queries to queries specific to the data source.
Steps:

1. The connector developer specifies the data which he wants to make available to the City Cloud.
2. The developers implements the connector interface to make queries possible from the City Cloud to the data source.
3. The developer submits the connector on http://city-cloud.eu.
4. The maintainers check the connector for validity and inform the developer about the outcome. If it was successful, the data will be viewable in the data explorer on the website.

SUBMIT DATA TO THE SYSTEM

Goals: Submit data to the City Cloud.
Preconditions: The provider is logged in to the system.
Summary: The web interface or API get used by providers to submit data into the system.
Steps:

1. The provider goes to http://city-cloud.eu and presses ‘submit data’.
2. The system displays the different ways to submit data into the system.
3. The provider decides what data type the data he wants to submit is part of and submits the data.
4. The system validates and shows if the submitting process was successful.

USER INTERFACE

This framework provides backend functionality and therefore does not contain a lot of direct UI. Only relevant parts with a UI are the online management interfaces for developers. These will be integrated in the main City Cloud website later on.

SYSTEM ARCHITECTURE

APPENDIX C

SUBSYSTEM DECOMPOSITION

Within the core system a few subsystems can be identified. Every subsystem is responsible for one of the core tasks of the system.

DATA SERVICE

The data service is responsible for management of all data. Every request of both data storage and retrieval goes through the data service.

ODATA SERVICE

The OData service takes data from the Data Service and provides it as an OData web service. This web service will be exposed to developers to allow them to query data from their application.

BUILDING BLOCK SERVICE

The Building Block Service provides an environment to host the building blocks for the mobile part of our application. Developers can upload their building block through the web interface and it will then be run by the building block service. The service will take care of running the building block, providing authentication and access to the data through the data service.

CONNECTOR SERVICE

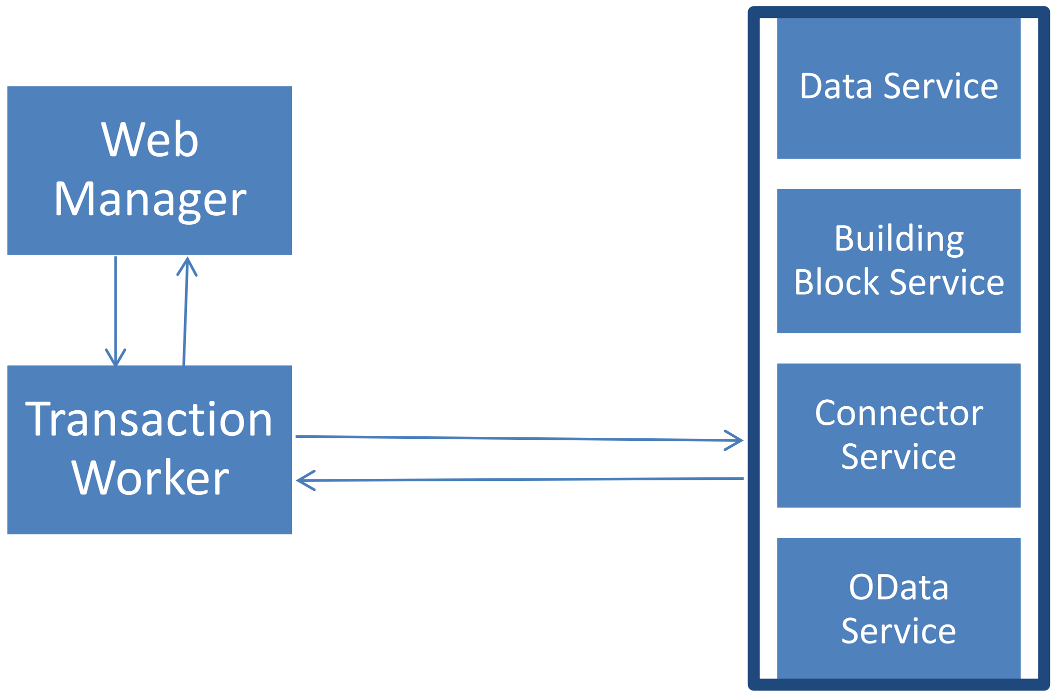
The Connector Service provides a way of connecting external data to the Data Service. Connectors can be uploaded through the Web Interface and will then be run by the Connector Service. The Connector Service supports two kinds of connectors. A connector can function as an interface to external data, only translating the queries, or it can function as a data parser which takes data from an external source and puts the data into our system's own data storage.

TRANSACTION WORKER

The transaction works handles all requests from the web manager to the components of the core cloud system.

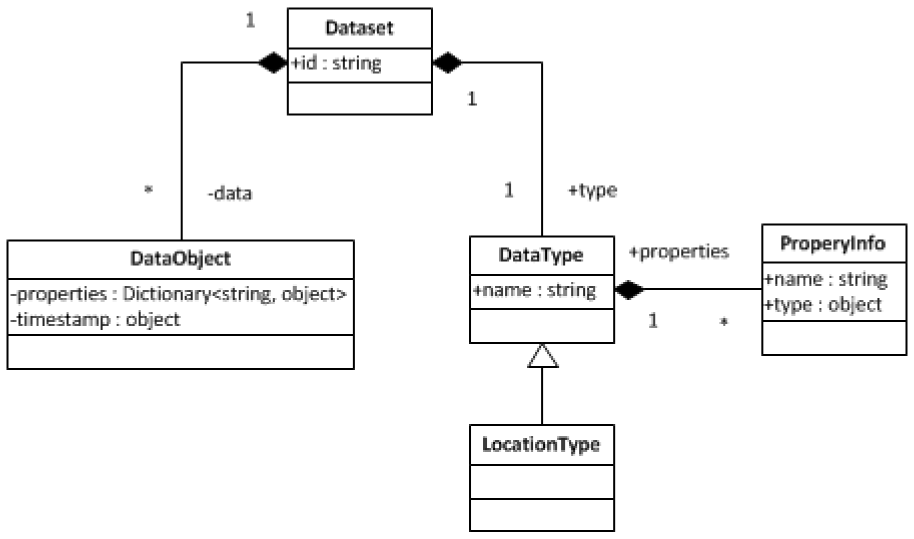
WEBMANAGER

The web manager provides all features required to manage the different parts of the core cloud system. Through a web interface both developers, data suppliers and system admins will be able to manage building blocks and connectors. They can also gain insight in usage data of different parts. All requests are sent to the Transaction Worker.



DATAMODEL

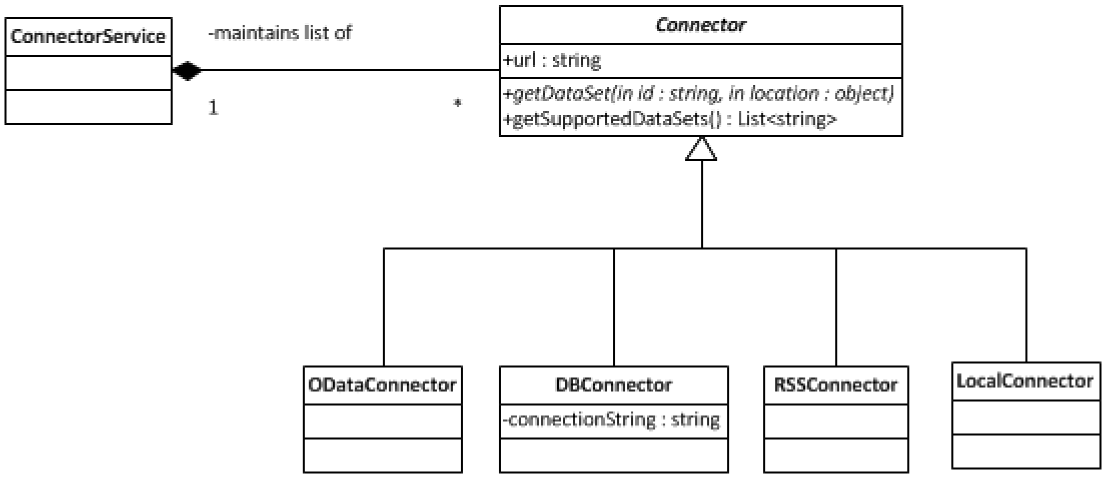
In Attachment 1 you can find the data model for the cloud service and the interaction with the cloud service. User interface classes are not taken into account in this model, because they are not part of the model. The model shows a high level overview of how the system will look like.



An important aspect is the design of the dynamic data types. The system works on Datasets. These datasets carry a list of generic data objects with specific properties. These data objects are described in a metadata class called a DataType. DataTypes can be delivered with a connector to extend the existing DataTypes and in that way the system can be extended with new DataTypes. An example can be LocationType.

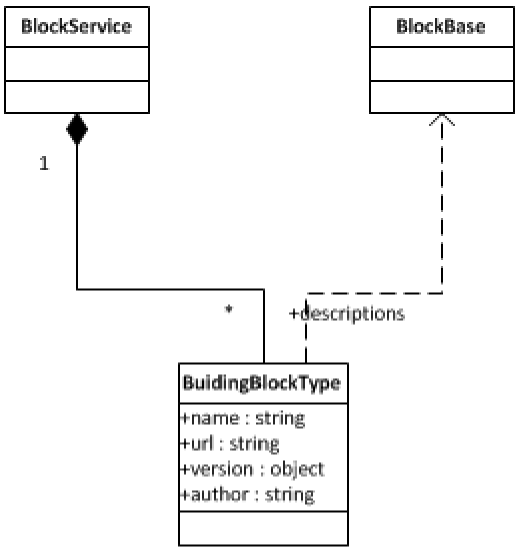
APPENDIX C

As described in the research document into this system, third party data services can be connected to our system via Connectors. The data model for the connector part is shown below.


ConnectorService
-maintains list of
1
*
Connector
+url : string
+getDataSet(in id : string, in location : object)
+getSupportedDataSets() : List<string>
ODataConnector
DBConnector
-connectionString : string
RSSConnector
LocalConnector

We will provide developers with a few base classes that give them a good start to work with our system. Every connector needs to inherit from connector base, which provides the underlying foundations for the other services to work with a Connector, like finding information on supported datasets and DataTypes. For data stored in our own system the LocalConnector is used. This is a special connector class that is not connected to third party data services, but just serves as a Table Data Gateway[1] to data stored in our own system. It is currently the only connector that supports write operations. Other connectors are read-only. It is not possible for developers to inherit from the LocalConnector class.

Another important part of the data model is the Building Block system as shown below.



The function of a building block is explained in the research document. Building blocks are represented in the system as a dataset of building blocks and can be used in the same way as other data types. A difference is that building blocks are managed by a special BlockService. A Building Block type describes a Building Block. It contains information on URI's, authors and versions.

[1] http://martinfowler.com/eaaCatalog/tableDataGateway.html

APPENDIX C

The OData web service will be a custom typed web service, based on a blogpost by Alex D. James, Program manager on the data services team on un-typed OData web services[2]. We will try to also support RDF and SparQL, techniques of the semantic web[3], in the web service.

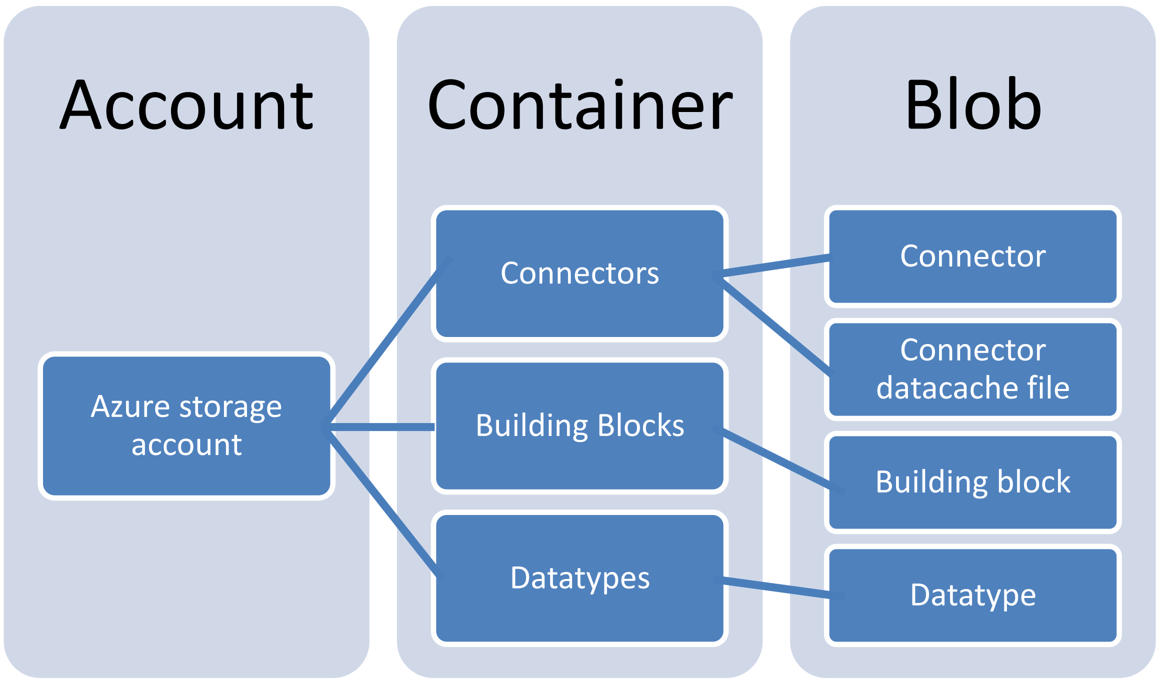
INTERACTION

To change, add or remove something in the system a transaction object needs to be built and executed. This is a typical form of the Command pattern[4]. These transactions will be added to a queue and executed when the thread is ready. This queue functions as an Active Object[5]. The relation between these transactions and the system as a whole can be seen in attachment 1.

PERSISTENT DATA MANAGEMENT

Blob storage:
Connectors (assembly)
Building blocks (assembly)
DataTypes (assembly)
Per connector cache

The system stores data in two of the three storage options of Windows Azure, Table Storage and Blob Storage. As explained in the research document Azure instances cannot be dynamically expanded with assemblies so we decided to store them in blob storage. Connector assemblies, building block assemblies and DataType assemblies are therefore stored in Blob storage. Additionally the data cache for every connector is also stored on blob storage. In this way connectors can improve their performance by caching data instead of do request to a third party data source every time.



Information other than binary large objects are stored in table storage. Table storage contains tables for information about building blocks, information about available connectors and information about DataTypes.

[2] http://blogs.msdn.com/b/alexj/archive/2010/03/02/creating-a-data-service-provider-part-9-un-typed.aspx
[3] http://www.w3.org/TR/rdf-sparql-query/
[4] Agile Principles Patterns and Practices in C#, Robert C Martin
[5] Agile Principles Patterns and Practices in C#, Robert C Martin

APPENDIX C

There will also be a table that holds information on statistics about requested data, used building blocks and more. As part of the platforms offer to store data in our own system there will be a table for every dataset that wants to make use of this service.

Table storage:
Building blocks Table
Connectors Table
DataTypes Table
Per dataset a table with data objects
Statistics table

ACCESS CONTROL

Access control is based on the types of users. End users, developers, providers and maintainers all have different levels of privileges. Privilege levels are handled primitively in the first version of City Cloud. The maintainers will provide developers and providers with the access token on their web page once they have logged in to the system, which they can use in their applications. The access control breakdown is as follows:

Everyone using the system, for both reading and submitting data, has to be logged in. Users can be elevated to developer status by maintainers through a request on the website. The access token string required in applications will become available after developer status has been granted.

Every connector and application which gets submitted will get verified by the maintainers in the first version. This is to prevent abuse.

TOOLS

Since the system pretty much relies on developer to build building blocks and connectors, tools are an important aspect of the system. The tools will exist of at least three assemblies:

- Assembly with the connector base class and helper classes. This assembly needs to be referenced to build new connectors.
- Assemblies with DataType classes. These assemblies are needed to add new DataType classes to the system. New DataTypes are added to the tools so developers can reuse them.
- Assembly containing base classes for building blocks. This assembly is needed to build building blocks for the phone application.

Besides the actual assemblies there also needs to be documentation on the API's and guidance on best practices. This will be a web application like many others for example the documentation on MSDN.

IMPLEMENTATION AND TESTPLAN

Since the system we are implementing is part of a dynamic Imagine Cup concept we need to be flexible during the whole implementation phase. Requirements may change and we need to be able to handle those changes.

APPENDIX C

To ensure that we work with the desired flexibility we will incorporate agile techniques into our process. We deliberately chose to cover only the high level system design and core functionality in this document, all the other parts will be defined when necessary. This is all part of the agile process.

To ensure the whole implementation process stay in line with the expectations, there will be regular meetings with all the parties involved. The core Imagine Cup team, including the people working on the concept, will have a weekly meeting to make sure we stay on the right track to the finals. Other involved parties like CapGemini, Microsoft and TU Delft will receive updates on a weekly basis. Feedback during those meeting and updates will be incorporated in the agile process.

The project is also subject to a strict deadline, the 7th of July, the day we are leaving for the Imagine Cup Worldwide Finals in New York. The one and only goal that really counts during the implementation is being able to show a working demo during the presentation at the finals to convince the jury. This requirement might contradict the desire to build a system in the most elegant and future proof way. This might mean that we need to sacrifice some elegancy to ensure a working demo can be delivered. This is also incorporated in the agile process.

During agile development it is important to make testing a key part of the process. Therefore test driven development will be used during all parts of the implementation. For every part of the system that is going to be implemented unit tests are written first. To facilitate this in a practical way we will use a Team Foundation Server with support for both building and testing the application. This will ensure that no code can be checked in before all tests are passed.

The Imagine Cup concept itself and the example cases that need to be presented during the presentation at the finals will be used as acceptance tests. The concept presentation provides a clear view of what the system needs to be able to do.

APPENDIX C

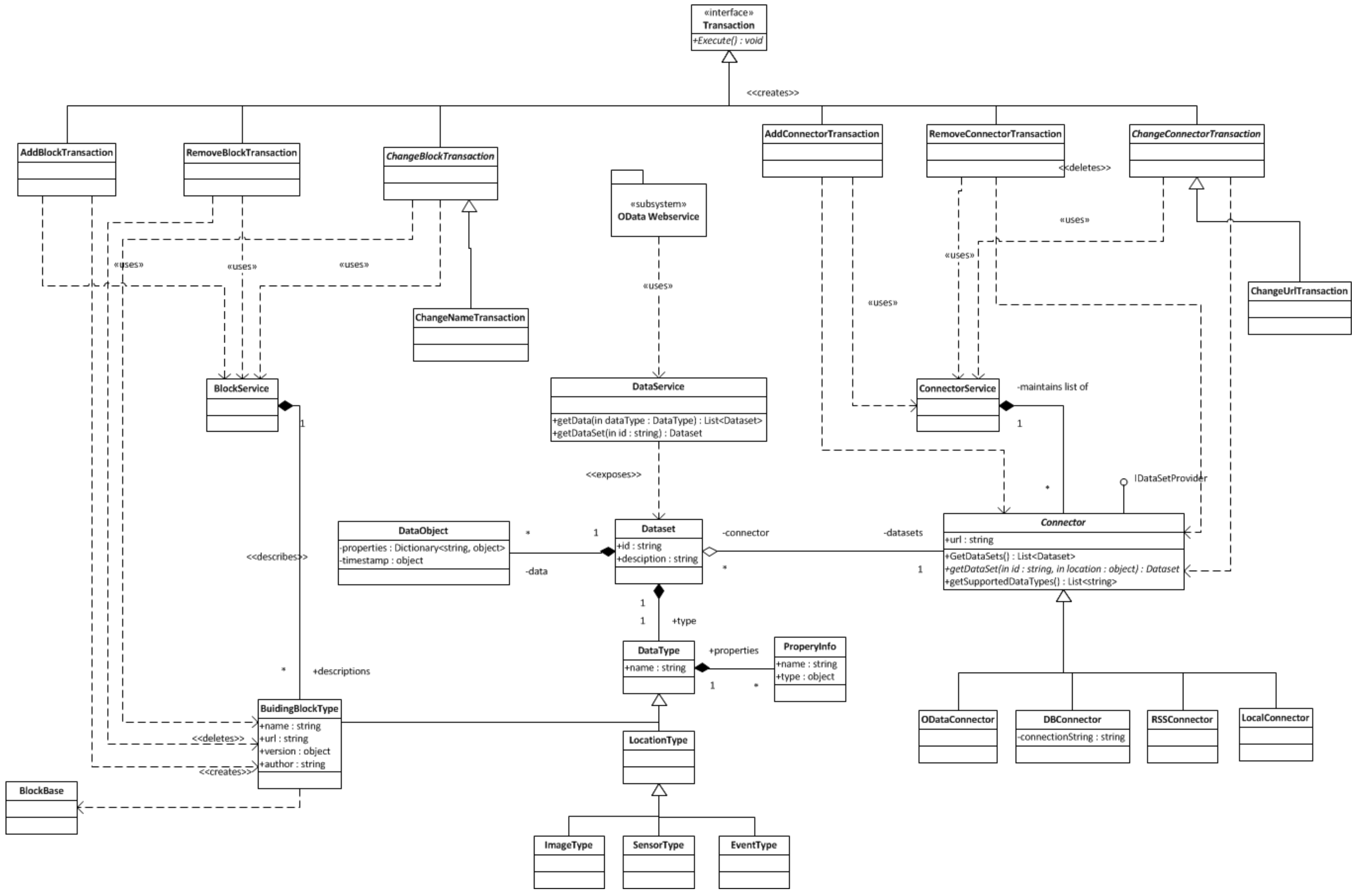


Figure 1. Old (now deprecated) class diagram