DELFT UNIVERSITY OF TECHNOLOGY
FACULTY OF EEMCS

AVANADE

Volta

Research and Showcase

Bachelor's Project IN3405

9/July/2008

Robin van den Berg	1224220
René Elstgeest	1096249
André Simões Dias Vieira	1263161
Erik Zuidema	1217798

Judging-committee

Principal	Drs. E. Jongsma
TU Delft supervisor	Dr. P.G. Kluit
BSc Coordinator	Ir. B.R. Sodoyer

THE LEADING INTEGRATOR OF MICROSOFT SOLUTIONS IN TODAY'S ENTERPRISE

Preface

This report is the result of the graduation project as conducted by students Robin van den Berg, René Elstgeest, André Simões Dias Vieira and Erik Zuidema, submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

The assignment was set up by Avanade as an internship from April to July 2008, to research the business and technical potential of the new technology: Volta. The assignment was approved as a Bachelor's project by TU Delft. Compared with Bachelor's projects of previous years, the focus on research has increased and is now an integral part of this report. It was a challenge to combine the different interests of Avanade, the TU Delft and simultaneously participating in the Imagine Cup.

An extra result of this project is the approval for writing an article for the September issue of *.NET Magazine*, a periodical by Microsoft for developers in the Benelux.

The writers owe many thanks to Gerardo de Geest for his support and guidance, Peter Kluit of TU Delft and Edwin Jongsma of Avanade for their advices. Also our gratitude goes out towards Student Society 'Christiaan Huygens', EESTEC LC Delft and Avanade for allowing us to use their resources. Furthermore, we would like to thank Albert Sluijter, Frans van Duijnen, Raul Kooter, Karen Hofman, Pieter de Bruin and Jan Drenthen for their creative input and Danny van Velzen for his technical support regarding Volta.

Table of Contents

Summary

Developing software as a service is a new trend in the IT industry. Volta is a new technology that targets simplifying this type of software. It offers developers a toolkit that enables Web development in any .NET language. This is done by recompiling the CIL code to JavaScript for the client tier (browser) and a .NET application for the Web server. The decision of determining on which tier certain code should run is easily carried through by adding an attribute to a class.

A showcase has been developed stretching the possibilities of the first *Community Technology Preview (CTP)* of Volta. A CTP is a pre-release version of software intended as a proof-of-concept for the community. The idea of the showcase was to provide users an impression of their carbon-footprint relative to others, accomplished by using Web services and information provided by users themselves. The experiences gained by building this showcase have been used in the research for Volta's value for Web application projects. The showcase was also submitted for Microsoft's Imagine Cup software development competition, which imposed additional requirements on the showcase. Many issues were encountered during the development, which forced the project group to adjust the requirements while implementing. The end result offers users login functionality, the ability to enter its food, house and travel CO_2-emission and provides an overview of the user's past and present carbon-footprint.

The project also focused on researching the possibilities of Volta. A number of research questions were posed and investigated in order to provide better information about the strengths and weaknesses of Volta. This led to the discovery of a number of problems from which the following suggestions for improvements were derived:

1. Provide more documentation
2. Equalize the behavior of Volta code in Debug and Release mode
3. Decrease the number of restrictions on classes
4. Increase cooperation with other technologies
5. Optimize performance
6. Increase security functionality

Although a developer should still be aware of the traps of distributed computing, Volta can offer benefits for developing applications.

- Web applications can be built with all .NET languages.
- Developers do not have to worry about the communication code between tiers.
- The facilities of the Visual Studio IDE can be used for testing, debugging and cooperating with other technologies.
- The end-to-end profiler is a useful tool for analyzing and optimizing a tier-split application.

The community technology preview of Volta looks promising. It is currently not fit for developing actual Web applications, but if the developers continue working on Volta and address the issues mentioned, Volta could prove to be a useful toolkit for Web development.

1. Introduction

Developing Software as a Service (SaaS) is a new trend in the software industry (1). Therefore, software companies like Microsoft are creating development tools which could make Web development easier. Software that is provided as a service often uses multiple tiers. This type of software primarily uses Web pages as a user interface. A high-level model of SaaS is provided in Figure 1.

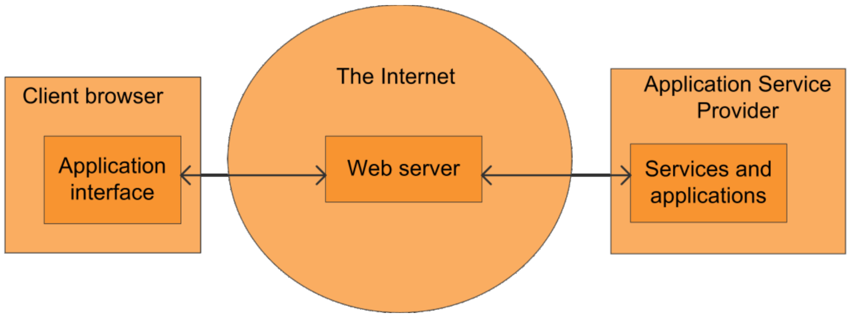


Figure 1. A model of how software is offered as a service.

The Web pages can be viewed in Web browsers, which make these applications platform independent. Usually a browser is provided on a client Operating System and is able to provide a user customized interface for logging in, offers shortcuts, etcetera. Using Asynchronous JavaScript And XML (Ajax) can increase the user-friendliness of Web pages, but usually requires developer-unfriendly JavaScript implementations (2; 3).

Development of multi-tier software introduces new challenges in comparison with single tier sequential software (4). A multi-tier application comes with the challenge of letting different programs and platforms communicate effectively and efficiently. An internet application often requires a database for data storage; it accesses Web services and needs an interface to communicate with the user. It would be preferable to be able to develop multi-tier applications without having to worry about the communication between tiers, and instead develop them as if they were a single tier application. Volta is a new technology that tries to achieve this (5).

This report focuses on answering the following research question about Volta:

What are the strengths and weaknesses of Volta for its meant purposes and how well does it cooperate with complementary and resembling technologies?

This project is being delivered by Avanade. For Avanade it is necessary to evaluate the usability of Volta for the development of their multi-tiered solutions.

To assess this, an actual application has been built using Volta. The experiences are used to answer part of the research questions. The application has also been submitted for the Microsoft Imagine Cup 2008. The theme of this software contest was "Imagine a world where technology enables a sustainable environment". The application is set up with this theme in mind.

Following this introduction there is a chapter describing Volta and a chapter describing other technologies important for this project. After these introducing chapters the three main parts of this report can be found. These parts are the Project, Showcase and Research parts and each end with an evaluation. The Project part describes the setup, planning and progression of this project. The Showcase part describes the implementation decisions and the details of the showcase. The Research part is a documentation of the research answering the questions of the pre-research. This pre-research can be found in appendix O. Following these parts a conclusion can be found, regarding the entire project and a Glossary explaining terms used during this report.

2. Volta

This report discusses the first Community Technology Preview (CTP) of Volta[1]. Volta's aim is to simplify the development of distributed applications. To achieve this, Volta provides developers the instruments to develop multi-tier applications as if they were single-tier applications.

This release mainly aims at proving easy tier-splitting is actually possible and enables the compilation of .NET languages to (X)HTML, CSS and JavaScript. This provides the possibility to create Web applications with Volta.

Volta acts as a recompiler: using Microsoft's compiler, C# is converted to Microsoft's Common Intermediate Language, CIL. CIL is a low-level programming language resembling assembly to which all .NET languages are compiled. Using a `[RunAtOrigin()]` attribute Volta is able to determine which code should run on one tier, and which on another. Volta comes with two compile configurations: *Debug* mode and *Release* mode. It is important to understand the difference between these two configurations in order to understand certain parts in this report.

In Debug mode, the communication will take place between two CIL executables, both running on the same computer. This does not introduce many restrictions. Communication between these two tiers can be compared to calling a library function from another program. The Debug mode is meant for assessing the correctness of the program logic. Compiling in this mode will not result in an actual deployable website, since a browser cannot execute CIL code. Figure 2 shows how compilation takes place in Debug mode.

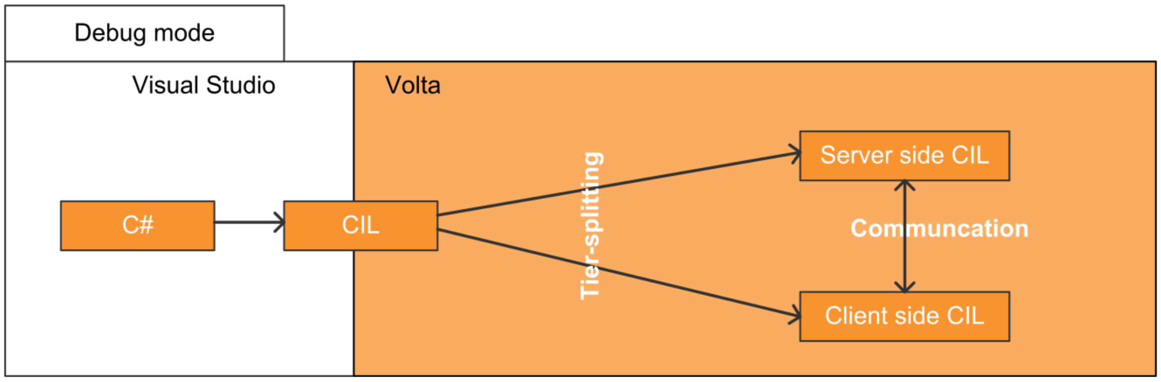


Figure 2. An overview of compiling in Debug mode.

In Release mode, the client-side CIL is actually converted to JavaScript. Since the CIL code is similar to assembly, the program structure of the original C# code is lost. The generated JavaScript will be based on the CIL operations without any optimization. Also, the system and library functions that are not compiled to CIL, but called as a subroutine cannot be included in the actual JavaScript, resulting in a *not supported* compile error. The communication between the server and client is done via HTTP Requests. Figure 3 provides an overview of how Volta compiles in Release mode.

[1] CTP-release of 5th of December 2008 with the bug fixes of 21st of February 2008.

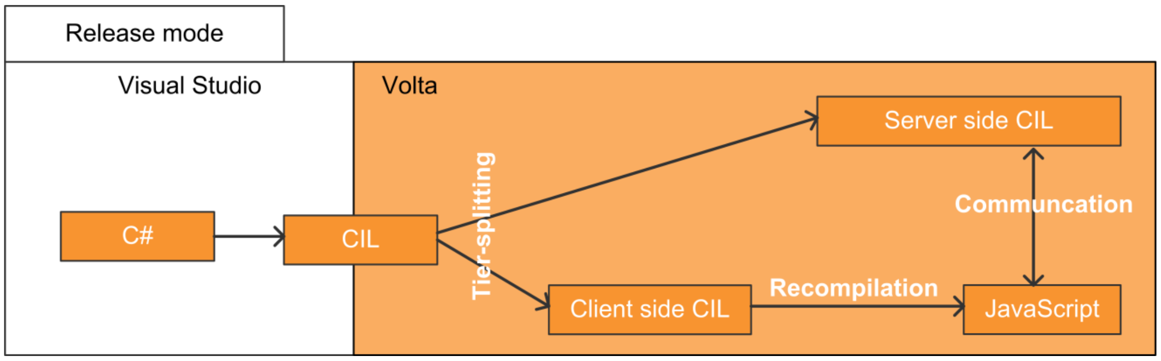


Figure 3. An overview of compiling in Release mode.

To be able to develop with Volta, Visual Studio 2008 should be extended with a Volta installer-file[2]. This installation configures the Debug and Release mode, and also provides an *end-to-end profiler*, called *Rotunda*. This profiler tracks the sequence of an application and shows the communication between tiers.

[2] The Volta installer file can be found at http://labs.live.com/volta/download/

3. Related technologies

This chapter explains technologies important to this project, because these technologies were used during this project and are mentioned throughout this report.

3.1. Visual Studio 2008

Visual Studio is an IDE from Microsoft, which can be used to develop applications in several programming languages. A Volta application requires this IDE for its development and compilation.

Visual Studio can use *Team Foundation Server*: a version control and project organization platform, and *IntelliSense*, which provides auto-completion and shows documentation. The provided *Object browser* is a good tool for searching methods and objects. Integration of WCF Web services is as easy as calling methods from a local library.

To create an application in Visual Studio, a *solution* is set up, which consists of several *projects*. Classes have *namespaces*, which resemble *packages* in Java. However, namespaces do not restrict classes to the same physical directory.

3.2. Silverlight 2.0

Silverlight provides specialized user interfaces that can be embedded in websites. A Volta application can look more attractive when using this technology since it would not be limited to the standard HTML interface elements.

Browsers require a plug-in which displays the interface and executes the program logic. A Silverlight user interface is defined in a specialized XML-based language called XAML. The interaction and logic is defined in a *code-behind* file which can be written in any .NET language. The browser plug-in can execute the CIL code of this code-behind file. The Silverlight application can access and be accessed by JavaScript. Silverlight has an advantage over the comparable Flash technology, since it allows automatic searching and indexing by Web crawlers.

3.3. Windows Communication Foundation

Windows Communication Foundation is a framework built by Microsoft to enable easy communication between applications. It has been part of the .NET framework since version 3.0. If an application provides a WCF-compatible service, a .NET application can easily connect to it and communicate with it via an optimized binary format (6). In the Visual Studio IDE, the WCF Web service functions can be used as if they were library functions. WCF implements WS-Addressing, WS-ReliableMessaging and WS-Security which are Web services (WS) standards for exchanging addressing information, assuring reliability and security.

3.4. ASP.NET

ASP.NET (Active Server Pages in .NET) is a programming language meant specifically for websites. It provides a method for including .NET code (e.g. C#) in a webpage, which enables a developer to build dynamic websites and Web services. The .NET code can be included into the regular HTML code of a webpage in an *aspx*-file, but is usually separated from this by using a separate *code-behind* file.

When a client's browser requests an aspx-webpage, the Web server will execute the .NET code in the aspx-file or the code-behind file. This will generate a webpage with the required content, which will be sent back to the client's browser.

This is different from how Volta builds up a webpage. Here, a regular html-file with JavaScript will be sent to the browser. This browser will execute the JavaScript which results in displaying the contents of the page. The JavaScript is executed client-side *after* the page is sent to the browser, whereas the .NET code of an ASP.NET page is executed server-side *before* the page is sent.

3.5. Virtual Earth

Virtual Earth is a Web application which can provide maps and data of user-specified locations in the world. Functions like determining the distance between two points are also available. A Web application can use these functions via JavaScript.

For Volta, a special Virtual Earth library is available which provides C#-methods for integrating the features of Virtual Earth into a Volta webpage. This means developers do not have to use a different programming language for this functionality.

Part I: Project

4. Introduction

In this part the project is described. First the principals are introduced, followed by an explanation of the assignment. In the assignment section the two different assignment parts are described: the research and showcase. The planning is then discussed followed by a description of the project progress. Finally an evaluation is given corresponding to the project organization, planning and progression.

5. Principals

The project was set up as an assignment of three principles. This chapter explains the interests of these principles.

Avanade

Avanade is a global IT consultancy company dedicated to using the Microsoft platform to help enterprises achieve profitable growth (7). Avanade is owned by Microsoft and Accenture. This way they try to benefit from the two strengths of consultancy (Accenture) and technology (Microsoft) knowledge and experience. With the help of Gerardo de Geest, Edwin Jongsma from Avanade supervised the project.

For Avanade, a technology like Volta could provide useful business benefits. Many solutions Avanade offers are multi-tiered. Consequently, it is necessary for Avanade to evaluate the usability of Volta for the development of such solutions. Speeding up development of multi-tiered applications could induce faster delivery of business solutions. Also, the cooperation of Volta with other Microsoft technologies is a valuable aspect for Avanade. This is because Avanade could then use Volta in combination with their existing solutions.

Delft University of Technology

This Bachelor's project is conducted at the faculty of EEMCS[3] at the Delft University of Technology in the Netherlands. Peter Kluit supervised the project, while Bernard Sodoyer was the Bachelor project coordinator for the TU Delft.

The TU Delft demands the ability to design and implement a software product from its Bachelor students. Besides this, research qualities are of increasing importance to the TU Delft.

Imagine Cup

The Imagine Cup is a yearly competition organized by Microsoft. This year's theme was "*Imagine a world where Information Technology enables a sustainable environment*". The showcase was submitted for the track of Software Development.

A jury will assess the application. They are primarily interested in how the application would enable a more sustainable environment. This will not be limited to the features, but the jury will also consider the appearance of the application and the impact it could have on the society.

[3] EEMCS: Electrical Engineering, applied Mathematics, Computer Science

6. Assignment

This chapter describes the assignment as posed by Avanade, the Delft University of Technology and the participation in the Imagine Cup. Section 6.1 provides a short background of the problem for Avanade and the problem description for our project. Formulating this problem helped us maintain focus throughout the project. Sections 6.2 and 6.3 explain the two parts of the assignment: the research and showcase.

6.1. Problem definition

Development of Web applications is difficult for large projects. A new technology like Volta could simplify the development, and offer a lot of freedom for designing multi-tiered applications. The problem is that up until the start of this project, it was unknown to the involved parties how Volta could be used for developing Internet applications and how it cooperates with other (existing) technologies.

The goal of the project is to research Volta's usefulness on tier-splitting and the ease of Web development and build a showcase application which displays Volta's possibilities.

6.2. Research

In the pre-research report, the concept of Volta has been investigated. This resulted in the formulation of the main research question:

What are the strengths and weaknesses of Volta for its meant purposes and how well does it cooperate with complementary and resembling technologies?

This question was split up in the following sub-questions:

1. Which possibilities and restrictions does Volta introduce with its tier-splitting?
2. What is the performance of Volta?
3. How does Volta contribute to secure development?
4. How does Volta cooperate with other technologies?
5. How does Volta perform in comparison with resembling and competing technologies?

The pre-research report, found in appendix O provided the line of thought concerning these questions. The actual research report can be found in Part 3. Using this research report, Avanade should be able to assess the value of Volta for their projects.

6.3. Showcase

Avanade required a showcase of the boundaries of Volta. Furthermore, the project team intended to extract the experiences gained from building this showcase to supply answers to the research questions. Concurrently, the showcase was submitted for the 2008 Imagine Cup. This means it had to comply with the theme of sustainability and the rules of the competition. Competing in the Imagine Cup enhanced the drive to create an appealing showcase. Due to our participation in the Imagine Cup, Avanade allowed changing the original idea of an Airport Control System to the *GReenGRasp* tool. The idea of this tool can be found in appendix J and the showcase is described in Part 2.

7. Planning

In this chapter the initial setup for this project is described. The planning and roles are described in section 7.1. Section 7.2 describes the problems that affected the planning and how they were dealt with.

7.1. Project setup

While setting up the first planning (appendix K) for the project the wishes of the three principals had to be accounted for. This planning had two initial targets: the research and the showcase implementation. Because the expectation was that these could be combined with participating in the Imagine Cup, the third target was making the application suitable for this contest. The planning was based on a number of assumptions:

- Volta allows for easy and fast Web development;
- Volta is in an early stage of development and might contain some bugs, however this should not pose too many problems;
- The project team would progress to the Dutch finals of the Imagine Cup.

To assure the deadlines of the planning were made, responsibilities were divided as follows:

• Project manager:	René
• Designer:	Erik
• Report coordinator:	Robin
• Test coordinator:	André

Furthermore, meetings were held every week to assure all team members were informed about each other's results. For these meetings pre-defined roles were assigned:

• Chairman:	René
• Secretary:	André

7.2. Project Progression

The initial planning mentioned above underwent two major changes worth mentioning due to unexpected complications. These complications were mainly caused by the following:

- lack of documentation of Volta;
- different behavior between Debug and Release mode;
- extensive restrictions imposed by Volta;
- data loss and delay in development due to repeated crashes of the Web server;

The lack of documentation and extensive restrictions imposed by Volta were responsible for the frequent changes in the design. This slowed down development. Furthermore, the different behavior between Debug and Release mode slowed down debugging. Part 2 discusses how this influenced the result for the application; Part 3 provides further details about the issues of Volta. Our main challenge was coping with the lack of time in combination with the deadline of the Imagine Cup Dutch Finals.

This was dealt with by changing the plans twice. The first change meant totally consuming the planned time buffer after only one month. At the same time the focus shifted to implementing features for the showcase for the Imagine Cup. This meant the research decreased in priority for a period of almost three weeks. This was to be increased again shortly after this period, in order to guarantee the quality of the research.

The second change implied splitting up the project team into two separate teams: Robin and René focused on the research and Erik and André on implementing the rest of the application. This was done so that each team member would only be required to concentrate on a single task. For the research part, Robin and René divided the sub questions amongst each other. Towards the end André and Erik split up the implementation tasks between user interface and business logic. Erik was responsible for designing and implementing the Silverlight interface, CSS and HTML elements, while André implemented the Silverlight to Volta communication and database interface.

8. Evaluation

Looking back on the project, complications required adjusting the expectations. This chapter discusses the lessons learned and also offers a retrospect for the positive elements of the project organization.

8.1. Planning

Working with an early release of a technology can result in many unexpected situations. A ten-week project of this nature with a buffer of only one week does not sufficiently account for uncertainties. More time should be reserved for set-backs due to bugs and undocumented issues.

When developing using a new technology time also has to be planned for familiarization. This enables a more realistic estimation the development time of features. Even though time for familiarizing was planned in the pre-research report of this project, it was mainly spent in studying the theory of Volta. It would have been better to spend more time on hands-on experience. In this project, the estimated development time was too optimistic.

System administration requires time, since a lot of issues normally arise in this area. This should be accounted for if these tasks are assigned to the project team. Although the project team was responsible for its own Web server, no time was planned for its administration.

8.2. Multiple principals

Combining the Imagine Cup with the requirements of the TU Delft and Avanade was not always easy, since these had conflicting goals. A lot of time went into developing a new concept for the showcase which could have been saved by using the idea proposed by Avanade instead. The new concept however did provide better opportunities to implement features that were useful for the research.

8.3. Project roles and cooperation

Splitting up the project team into a research and implementation subgroup worked well. It gave the members the possibility to concentrate on a single task. This somewhat relieved the problems of the duality of research on one hand and software design on the other.

The implementation tasks were divided naturally between André and Erik. This worked out well, even though sometimes one had to wait for the other. In these cases pair programming was applied to use the time of the waiting team member usefully.

The division of the research part into sub questions helped with planning this aspect of the project. However, the actual questions sometimes overlapped, which made it difficult to prevent both team members from researching the same aspects. This was primarily avoided through good communication.

Because the team members worked in the same room, communication during the project period was very effective. Furthermore, the regular meetings with the project team and supervisors helped with making difficult decisions. The minutes of these meetings were a convenient documentation of the decisions of the team.

Part II: Showcase

9. Introduction

This part describes the implementation of the showcase for this project. First, the specific assignment for the showcase will be described. This is followed by the requirements analysis. Chapter 12 provides a general description of our showcase, including complicated solutions that were implemented. The specific interoperability workarounds can be found in chapter 13. A brief section follows describing the Quality Assurance. This part ends with an evaluation of the implementation track.

10. Project assignment

One part of the assignment was to build an application exploring the boundaries of Volta. Originally this showcase would have been an Airport Control System. Because of our participation in the Imagine Cup, Avanade allowed us to come up with our own idea. The new idea, *GReenGRasp*, provided more links with our research, since the airport showcase would only implement the cooperation with Virtual Earth.

The goal of GReenGRasp is to provide users with a framework to monitor and stimulate the reduction of their personal carbon footprint. More information can be found in appendix J. The technical goals were to build a website with a database backend using Volta and its tier-splitting capabilities. Furthermore, cooperation with Virtual Earth was to be implemented. In a later stage, cooperation with Silverlight was introduced to provide a rich user interface. Finally, Windows Live ID was introduced using ASP.NET next to Volta.

11. Requirements analysis

The initial set of requirements was created based upon the original idea for the application. This was a feature rich application. These features were divided in a MoSCoW-analysis. However, due to the complications mentioned in section 7.2, a decision was made to build a showcase instead. These complications were in short:

- the lack of documentation of Volta;
- the different behavior between Debug and Release mode;
- the extensive restrictions imposed by Volta;
- the data loss and delay in development due to repeated crashes of the Web server;

This showcase had two functions. First it had to explore the boundaries of Volta, aiding the research in various questions. The second function was to serve as a demonstration of the GReenGRasp idea during the Imagine Cup. Due to the change of objectives most features described in the MoSCoW-analysis were dropped. Since these did not further explore the boundaries of Volta. An overview of the MoSCoW-analysis can be found in appendix D.

An overview of the requirements can be found in appendix C. This describes the original set of requirements with the new Silverlight feature. The crossed out features were not implemented in the showcase.

The implemented features in the showcase served the following purposes:

- Appealing graphical user interface;
- Testing performance of Volta;
- Testing general web development with Volta;
- Testing interoperability of Volta;
- Increasing chances at Imagine Cup.

The login, news, and mailing list were implemented to test general web development, performance and interoperability with MS SQL databases. The mailing list was however never integrated with the site, because Volta did not support the library functions needed to send emails from the server. A special login through Windows Live was implemented for the Imagine Cup and to test interoperability with ASP.NET.

The integration with Silverlight was implemented to satisfy multiple purposes. The communication between Volta and Silverlight helped test the interoperability. While the actual integration of Silveright into the showcase provided an appealing graphical user interface and increased our chances at the Imagine Cup.

Virtual Earth was integrated into our application to fulfill the assignment requirement set by Avanade. It also contributed to an appealing user interface and provided an appealing feature for the Imagine Cup.

CO_2-calculation and consumption of the Smart Metering Webservice was primarily added to demonstrate the GReenGRasp idea for the Imagine Cup. Furthermore, the consumption of this Web service was encouraged by Avanade, due to interest showed by one of their clients. To test interoperability WCF was used to consume this Webservice.

The implementation of these features forced the discovery of new complicated solutions and workarounds. For this reason, these can be found in chapters 12 and 13.

12. General System Description

The system built combines different technologies throughout both server and client tiers. A deployment diagram can be found in Figure 4. This diagram shows the connection with the database, the separation between browser and server packages, the different technologies used and which packages communicate with each other. IIS is used to host both the Volta Server and ASP.NET. Both Volta and ASP.NET communicate with the database through SQL and Volta uses WCF to consume the SmartMetering Web service. On the browser side Internet Explorer 7.0 must be used to open the Volta page and display the Silverlight 2 Beta 2 application used for the GUI.

A more detailed package description can be found in appendix E. It models the different packages for each tier specifying communication between classes, implementation of interfaces and inheritance.

Visual Studio generated class diagrams can be found in appendix F. Specific implementation details are described in the following subsections.

The following subsections describe how the database interface layer, components and layout were implemented. These implementation details were included in the report since they describe complicated constructions used in our showcase.

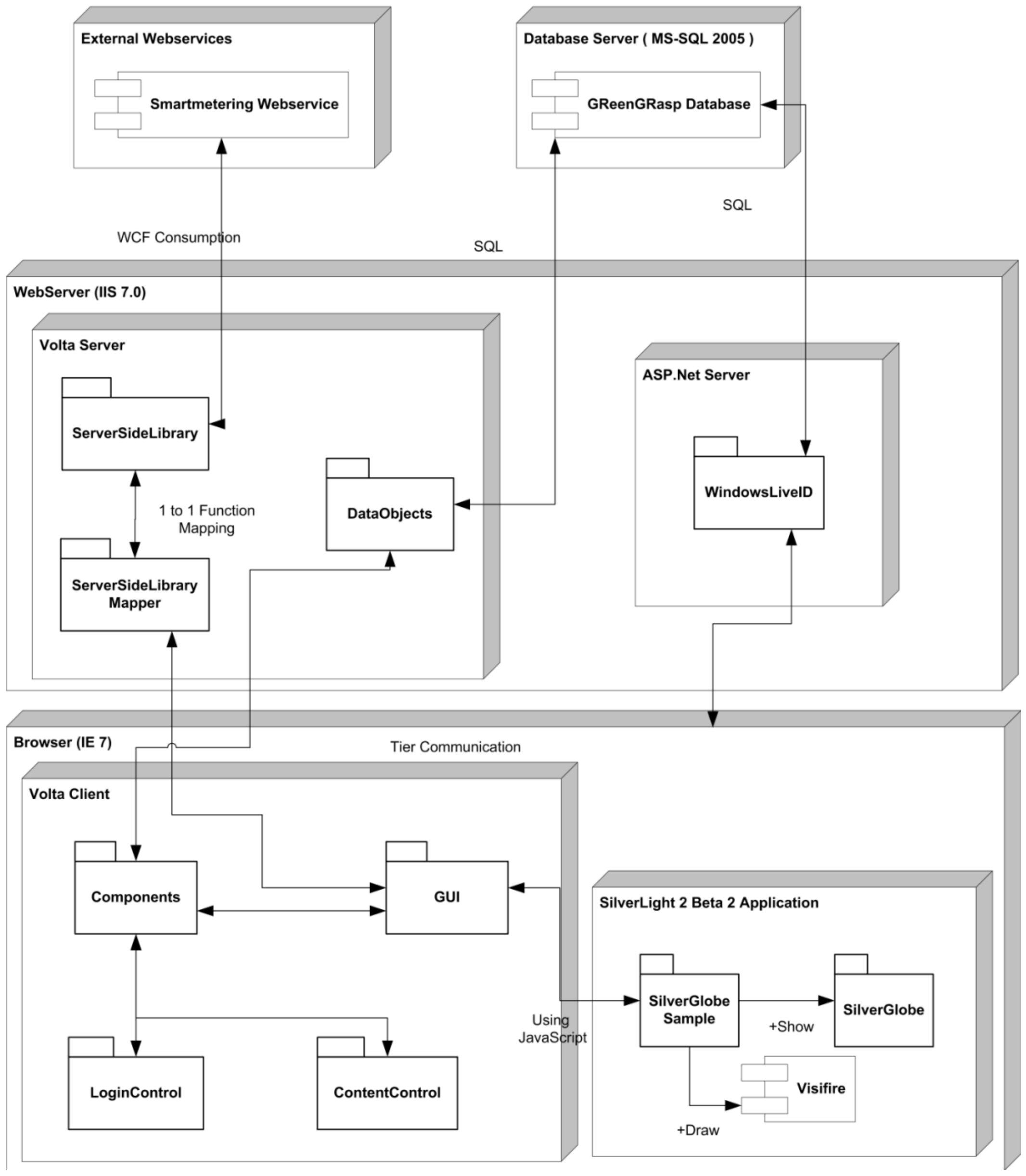


Figure 4. Deployment Diagram

12.1. Database Interface

The application uses a MS SQL database and a custom built interface layer of `DataObject` classes to communicate with the database. This `DataObject` is used as a replacement for *LINQ*. This `DataObject` implements the interface class `DataItem` which contains three properties: an *id*, a *Property list*, and a `DataConnection`. Each `DataItem` corresponds to a table in the database and contains a set of properties for each column. This set of properties consists of a property named identically to the column, a Boolean property to track changes for later updates. A property with a lower case first letter is added, which enables setting its attribute without triggering the `changed` Boolean.

The `DataConnection` is the class that contains the database connection. All the database operations take place in this class. A `DataConnection` class has a two-way reference to a `DataItem`. It is used by the *DataItem* and it uses its properties list mentioned above, to be able to reach its attributes. Basically the properties list is used here as a reflection substitute, since Volta does not support reflection. To write a `DataItem` to the database, the `DataConnection`'s *Create*-method first fills the column names. It uses the properties list and uses the same list to get the value of the corresponding properties through the `GetProperty(string PropertyName)` method of the C# `Type` class. The `DataConnection`'s *Update*-method uses the same principle, however it uses the additional Boolean property to check whether the column needs update.

The package diagram, individual class diagrams of the data objects and the database model diagram can be found in appendices F and G.

12.2. Components

The application is component based: basically a component can work by itself and has its own responsibilities. The components are able to communicate with other components or might depend on other components actions. The C# event framework was used to overcome the C# restriction of circular project references. Each component has events to which other components can register. The component fires its event after a certain action and all the components subscribed to it are notified. This is a variation to the observer and observable pattern. This component idea provides an easy way to build various blocks of the site and gives the flexibility of placing them wherever needed, see the Layout section for more details.

12.3. Layout

Most of the components have their own GUI part and there is one manager that places the component GUI parts on the page. That is why these two packages are so tightly linked. Every component extends from `Block` which implements the interface `iBlock`. Because of this, the `GUIManager` can be sure that every given component has the methods it needs to place the component on the site. The `GUIManager` knows the positions (HTML `Divs`) where it can place these blocks and decides if it is allowed to place the component there. For example it places the `LoginComponent` in our header and the `NewsComponent` in the main part of the site.

The main site is fluid, has bright colors and a colorful contrasting background in comparison to the white foreground. It has the standard column layout, a header with a logo, followed by the main menu bar. The main part has a block to the right for navigating through the whole site. The page ends in the bottom with a footer. The markup of the website has been implemented with HTML and CSS.

For a detailed description of the GReenGRasp Silverlight interface with screenshots see appendix H.

13. Interoperability workarounds

This section describes the workarounds used in the showcase to have Volta cooperate with Silverlight, WCF and ASP.NET. These workarounds were of great interest to the research for interoperability of Volta, see chapter 21.

13.1. Silverlight

Silverlight was implemented to provide an appealing graphical user interface. The Silverlight application is loaded inside a `div` of our Volta `page` after a user has logged in. Within the Silverlight application the user can enter its information regarding the three different GRasps, view their buddies, calculate and display their GReenGRasp. For displaying the GReenGRasp we used VisiFire, which is an open graph design package for Silverlight.

The communication between the Volta client and Silverlight application is done via the `GReenGRaspWebsite` and `SilverLightCommunication` classes on the Volta side and via `SilverGlobeSample` on the Silverlight side.

All communication to and from Silverlight uses JavaScript. For a detailed description of how this communication was implemented see appendix N.

13.2. WCF

Adding a WCF class to a Volta Application will not work. This is due to the fact that it can neither be recompiled to JavaScript since it is not supported, nor can it be run server-side due to the server-side inheritance restriction, explained in section 18. At first working around this limitation by consuming the service using an XmlHttpRequest was the best option. Later, a workaround for the library compatibility issue was discovered. This workaround consists of placing the unsupported classes in a signed library. In the application we called this library the `ServerSideLibrary`. This library must be ignored by the Volta assembler through the `IgnoreAssembly` attribute and must only be used by a server class in order to work properly. For this reason we decided to create server-side mapping classes, enabling the use of the `ServerSideLibrary` functions on the client tier. These classes can be found in the `ServerSideMapping` package. Furthermore only *serializable* data can be sent from the `ServerSideLibrary` to the server class.

13.3. ASP.NET

ASP.NET does not really cooperate with Volta: ASP.NET gets processed and executed before Volta, so it runs next to Volta. The trick is to make two projects: an ASP.NET project and a Volta project. Copy the ASP.NET project in the folder of the Volta project and rename the Volta *Page.html* file to *default.aspx*. Then for publishing purposes, the code-behind files of ASP.NET must not be compiled in the Volta project and neither should the *default.aspx* in the ASP.NET project. When deploying, first publish the Volta project to a directory and then publish the ASP.NET project to the same directory. ASP.NET has to go last and it has to ignore the *default.aspx* file from being copied. Otherwise, it will overwrite the added JavaScript to initiate the Volta code. This cannot be set in the Volta project, because then the Volta code-behind would not be compiled.

13.4. Windows Live

A prerequisite for the Windows Live integration is to have ASP.NET working (see previous section). This is because the SDK delivered by Windows Live ID was available in several Web languages, but not in Volta-compatible C#[4]. Conversion of this C# code to Volta compatible code was abandoned due to too many unsupported libraries. The code in ASP.NET was pretty straightforward. A control displays a link that redirects the user to the sign-in page of Windows Live. From there, it makes a POST-call to the URL specified on a webpage of Windows Live. On a different website you are able to manage your websites that

[4] During the implementation period of this project.

use the Live login. On this website you can retrieve a key, necessary to use the login feature for your website.

After signing in, the user is redirected back to our webpage. There, the ASP.NET handles the POST-data sent along with the redirection. The POST-data was also a problem at first, because a Volta HTML page cannot process it. This is because JavaScript has no access to its own HTTP-request and therefore cannot access POST and GET variables. ASP.NET *can* access this, and places the token acquired from Windows Live in our database on the server side. Subsequently, this is passed to the client side via a hidden textbox. With this construction, a user is still logged in when he or she refreshes the browser.

14. Quality Assurance

Testing was performed manually and very superficially during the implementation of this showcase. The decision against automated or extensive testing was based on two facts. First the C# code of Volta can only be tested in Debug Mode, since in Release Mode the code is recompiled to HTML and JavaScript on the browser side. Furthermore, the behavior in Debug Mode does not resemble the behavior in Release Mode, as explained in chapter 2 and section 17.2. Secondly, there are no tools available to test the recompiled Volta code, which is explained in section Q 2.6 of chapter 19.

Instead the focus shifted towards providing correct documentation of the limitations and flaws of Volta, for which a result can be found in part 3. Furthermore, the team attempted to acquire as much information as possible regarding the current possibilities of Volta by being active on the Volta forums.

15. Evaluation

This showcase was used throughout this project mainly as a tool to explore the boundaries of Volta. This showcase should not be seen as a demonstration of what can easily be achieved with Volta. Many of the features were implemented using complicated workarounds. However, the exploration of the boundaries of Volta using this showcase provided the research with useful information.

During the implementation of this showcase our team was active on the Volta development forums. Mostly inquiring about limitations or bugs, however, some used solutions were posted as well. Furthermore, this showcase can show the Volta developers how Volta can be used in its current form. Together with the research, presented in the following part, the Volta developers can take advantage from our work, since these highlight some of the limitations and flaws of Volta.

Part III: Research

16. Introduction

In this chapter, the results for the different research items are discussed. Each main question consists of multiple sub questions. Some of these questions are combined with one of more other questions due to a large overlap in the answer. This is indicated at the questions where this is applicable.

16.1. Research aims

At the start of the research a main research question was set up:

What are the strengths and weaknesses of Volta for its meant purposes and how well does it cooperate with complementary and resembling technologies?

This is a very general question that cannot be answered with one simple answer. In order to go into enough detail and to keep a good overview this question is separated into five main questions:

1. *Which possibilities and restrictions does Volta introduce with its tier splitting?*
2a. *What is the runtime performance of Volta?*
2b. *What is the development performance of Volta?*
3. *How does Volta contribute to secure development?*
4. *How does Volta cooperate with other technologies?*
5. *How does Volta perform in comparison with resembling and competing technologies?*

Every question is separated into their own sub questions. This way the research looks as a tree with the main question as its root and two other layers as shown in Figure 5.

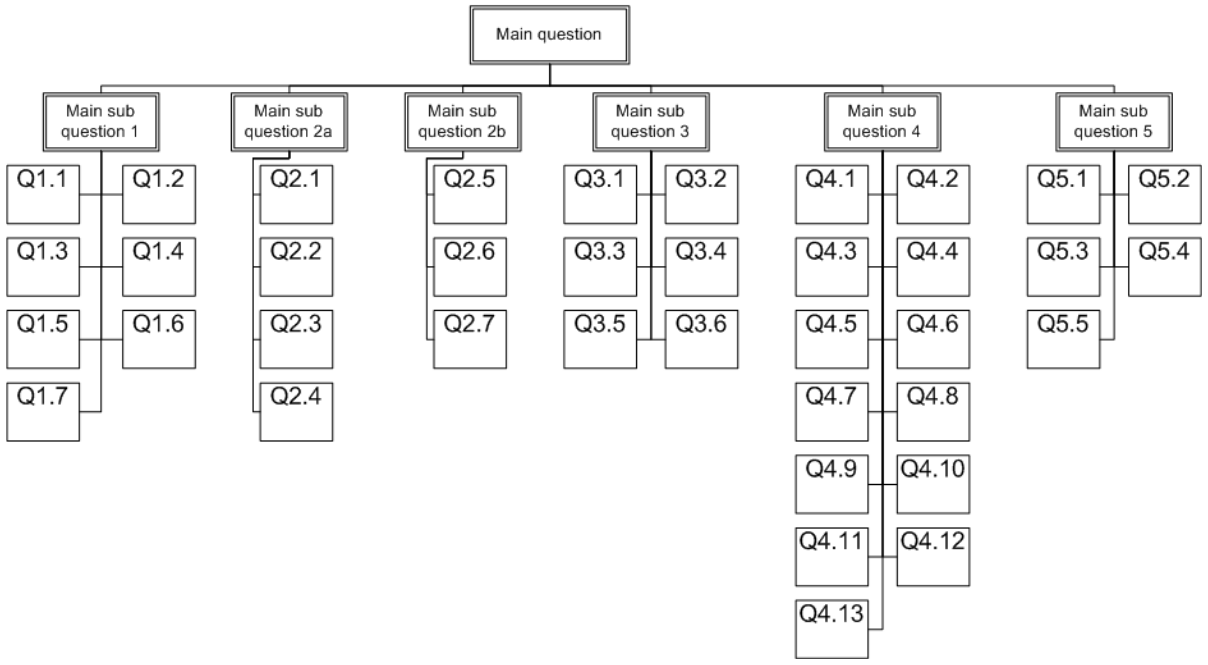


Figure 5. Research structure

16.2. Notes on solution applicability

The questions of this research were answered based on the experiences obtained by implementing the showcase discussed in Part 2 and additional research. However, due to the lack of documentation of Volta and the sometimes unexpected behavior we cannot guarantee a total accuracy of our findings regarding the absence of features or existence of faults. The provided samples worked for the explained scenarios and are designed to work in general. However, errors due to unexpected behavior of the framework are not accounted for when these samples are used after modification.

17. Overall findings of Volta

Throughout the research there were a number of general findings that do not fit within a specific question. These subjects are discussed in this chapter. These overall findings should be kept in mind when reading the answers of the research questions.

17.1. Absence of documentation

A major drawback of this Volta release is that it is not shipped with a detailed documentation on how the system works. The website of Live labs does offer recipes that show how to use Volta in some situations, but it does not cover nearly enough to verify what Volta is or should be able to do. The site does explain the vision of Volta and explains on a very abstract level what Volta is able to do. However, some development limitations were discovered that would not be expected from Volta when just examining the website.

Documentation could easily be provided in a possible future version. The expectation is that it will increase the usability of Volta significantly.

17.2. Debug and Release mode

The Volta plug-in for Visual Studio comes with two separate configuration modes for the developer. Debug mode comes with the usual debugging facilities, but does not represent the actual functioning of Volta. For this reason, constructions that work in Debug mode might possibly not to work in Release mode. Release mode does represent the actual working of Volta, which gives a more realistic look on how Volta works, but does not allow any tools for debugging.

One example of the differences between Debug and Release mode is that the runtime performance in the browser is significantly influenced. Throughout this research the results are mainly based on Release mode development and sometimes even published applications are used (or even needed). For more information on the fundamental difference between Debug and Release mode please take a look at section 2.

It is unclear if the difference between Debug and Release mode will change in future releases. Generating errors instead of warnings when compiling not supported methods in Debug mode is a first option that would significantly decrease the difference.

17.3. Tier-splitting with Volta

With this CTP Volta has shown that automatic tier-splitting is possible. This does not mean that Volta solves the fact that not every tier is suitable for any situation. For example, the browser is less capable to calculate tough mathematical calculations which are more likely to run on a specialized machine for calculation intensive applications. A browser however, is very suitable to present a GUI that targets multiple platforms. Volta will not remove the need for this kind of decision making. Another thing Volta does not solve, are the challenges that shared variables introduce when computing is distributed among more than one process. Also running performance of currently generated JavaScript by Volta for the browser is not suitable for real life application development. More details about this will be answered throughout this research part.

17.4. Other features

Clearly this CTP is far from ready for release judging on the support of the .NET framework's library functions. Quite a lot of library functions are simply *not supported*. This is probably mainly due to some implementation issues that are still needed to be solved in Volta. For example, inheritance is unavailable for classes in the server tier. It is possible to solve this problem with a workaround, but these are not very developer friendly.

18. Main question 1: What possibilities and restrictions does Volta introduce with its tier splitting?

Tier-splitting is the major issue that Volta addresses. Its developers present a way to create a single tier program and split this into multiple parts that can run on different systems. This chapter will go deeper into what advantages this brings and what restrictions it introduces if Volta is used for a project.

Q 1.1 What advantages does Volta create for tier-splitting?

Volta enables a basic possibility to actually split a program to run in different tiers without having to think about the communication code between the tiers. This should generally simplify the development of a multi-tier application. Volta shows it is possible to split up a program by choosing an execution tier for each class separately at the end of the development track. This tier-splitting does have hidden limitations mainly due to limited support of .NET library functionality.

Writing program code

As long as a developer keeps in mind the limitations Volta currently has, a multi-tiered application can be developed as if it runs on a single tier. The decision of where each class will have to run can actually be made after the conventional development cycles. By entering `[RunAtOrigin()]`[5] above the class code a developer instructs the compiler to compile the class to the server tier. Currently, Volta supports only two tiers: the (Web) server tier and the browser (client) tier.

The server tier

Server-side classes run on the server, obviously. This implies that such classes can access server-side classes in the normal fashion. One has to keep in mind that the main application works from browser to server tier, since the browser runs the user interface. This means the main control of an application lies in the browser tier and the server-side processes are invoked by the browser. A big limitation of the server tier is that it does not support inheritance. This can be worked around with a workaround used in the showcase described in section Q 1.2.

The browser tier

Programming for the browser is limited, especially regarding communication from the server tier towards the browser and some classes that cannot be compiled to JavaScript. This will be further explained in sub question 2.

Volta enables the use of the browser integrally within the development. It makes use of the Document Object Model (DOM) to implement dynamic Web applications. The generated code therefore implements a standard that is supported by the major browsers[6].

Q 1.2 What complications does tier-splitting in Volta bring?

The Volta developer team published the following list of *known issues and limitations* (8):

1. Only subclasses of `Object` can be tier split.
2. The serialization between the two tiers is not very fast.

[5] Future releases should be able to handle `[RunAt("TierName")]` attributes.

[6] Not every browser has been tested for Volta compliance and not every browser complies with Web standards.

3. The Visual Studio support for tier splitting supports only one server.
4. The server URI can only be specified as an attribute argument (i.e., in source code). Consequently deploying to a different server requires recompiling the code.

The first issue is a substantial problem. Inheritance is not supported for server-side classes, so all library classes and functions need to be run client-side. This can be a problem for handling database connections, for example. There is a workaround for this limitation. First, the classes needed on the server should be compiled in another project to a signed assembly. This makes it possible to include these files as a reference. By telling the Volta compiler not to recompile those references and only reference these from classes that do run server-side enables usage of classes that make use of inheritance. Of course, this is not an optimal solution. This workaround is applied to be able to use the WCF-functions for the SmartMetering Web service in the GReenGRasp showcase described in section 13.2.

The second issue is discussed in question Q 2.1.

The third issue limits the research to comparing only running the code client-side versus running it server-side. It is currently not possible to split Volta-websites across a group of (load-balanced) Web servers.

The fourth comes from the fact that Volta does not support configuration-files. These files come in handy when defining common variables, such as a database connection string. Not being able to use these means these variables must be declared somewhere in the source code and passed along, or in multiple locations, which makes changing the values difficult.

Also, some other classes of the Base Class Library are *not supported*. This library should be accessible for every .NET language. This means that the Volta compiler is not able to compile these classes (and its functions) to JavaScript. To work around this, a helper-class can be created at the server side, which executes the necessary methods and returns the results to the client. An example is `System.DateTime`, which is used in the GRasp-methods (see Code sample 1). This construction results in extra server-calls, which increases response time.

```
namespace GReenGRasp
{
    [RunAtOrigin]
    class DateTimeHelperClass
    {
        public static TimeSpan subtract(DateTime start, DateTime end)
        {
            return end.Subtract(start);
        }

        public static int dayOfWeek(DateTime date)
        {
            return (int)date.DayOfWeek;
        }
    }
}
```

Code sample 1. Server-side helper class for DateTime functions

Another issue is the communication from the server to the client. Not all objects can be sent from server to client, and no (references to) HTML elements can be sent from client to server. This means if something happens on the server side, it cannot push notifications to the client. This seems obvious when you think about how the http-protocol is set up, but for regular (distributed) application development, it is unusual. It is possible to set a certain flag and let the client periodically poll this, but for a large application, the number of flags to poll can increase quickly, making this solution unsuitable.

On the Volta Web forum, a question was raised about whether it would be possible to use some kind of Comet[7]-style messaging system. Wes Dyer of the Volta development team responded with: *"In the first release of Volta, server-to-client calls are not allowed. Of course, it is entirely possible. We only had so much time for the first release. So look for it in some upcoming release."* (9)

Also, in *Release mode*, threading is not supported at the server side, so the server cannot actually do anything without a client's action. Another consequence of this is that asynchronous calls are handled sequentially.

In short, there are still many smaller issues that are needed to be solved in order to effectively simplify development in Volta. Technical limitations of the browser still pose problems that need to be solved in order to be fully able to split programs among tiers.

Q 1.3 How can Volta be used in Brownfield scenarios

In Brownfield scenario's issues arise where new software need to coexist with existing applications. This question is not thoroughly examined, but it there are two subjects worth mentioning.

The first advantage of deploying Volta is that it is able to run on IIS 6.0 and 7.0 and for situations where for example ASP.NET applications are already deployed it is not necessary to change the current architecture. Also the connection with MS SQL works fine. Deployment of new Volta application does not require particularly new software architecture to run on.

Connecting a Volta application with an existing ASP.NET application is also possible (to a certain extend) as described in section 13.3. The main limitation of this solution is that custom libraries are not easily accessible from Volta and is only possible at the server tier using the workaround describe in section 13.2.

Deploying a Volta application in existing Microsoft .NET oriented architectures is not difficult. Main question 4 discusses the cooperation with other technologies.

Q 1.4 What problems regarding shared data can be solved using Volta?

Although it is possible to set the visibility of classes and methods with C#, this does not stop you from running into shared data problems, nor is it a Volta-specific feature. C# lets you easily create *get*- and *set*-methods, which can help define critical sections, but the choice of implementing these restrictions is still in the hands of the programmer. Volta does not seem to offer tools to solve these issues, nor does it seem to proclaim it does: *"Volta doesn't eliminate the intellectual challenges of distributed computing. We must still formulate strategies for partitioning functionality and dealing with network latency and availability."*(8)

Switching a class from client to server-side using the `[RunAtOrigin()]` attribute can alter program behavior. When a class uses static variables, these will be shared among the clients when it runs on the server side, which means its value is uncertain. When the same class runs client-side though, every client has its own instance and the values will only be changed by this client. We have made a small chat-application that uses this property; see appendix A for the program code.

When tier-splitting is relatively easy to achieve, this can be a danger for safe programming. Since this means it is also easier to create shared data-problems for inexperienced programmers. However

[7] Comet programming is a generic term for web application techniques that use a long-held HTTP connection which allows a web server to push data to a browser (22).

interesting, the discussion about how tight or lose a programming language should be, is outside the focus of this research.

Q1.5 What timing problems will tier-splitting with Volta introduce?

Splitting up a single-tiered application can introduce timing problems which did not exist before. These timing problems are not new. These timing problems are the same for regular multi-threaded and distributed applications. A scenario where this would change the behavior of the application is when multiple clients use a shared variable as described in the previous section. In this scenario, it is possible to program something that would work in a single tier situation, but this introduces a timing problem when split up.

```
String localClientString = serverSideClass.getSharedVariable();
localClientString += “Appending a new line\n”;
// do something time consuming...
serverSideClass.setSharedVariable(localClientString);
```

Code sample 2. Code that will not run as you might expect

An example of this is Code sample 2. Here, it is possible that two or more client processes ‘append’ something to the shared variable and only the last process is successful. The solution provided in Code sample 3 results in correct behavior since the append-operation is now atomic and the server processes them in a set order.

```
// do something time consuming...
serverSideClass.SharedVariable.append( “Appending new line\n”);
```

Code sample 3. Code that will work in all tier situations.

There might be other cases where the shared variable is necessary for the correct working of the methods between receiving and sending of a variable. In these cases, it would be best to introduce a semaphore for the variable. This is not something one expects when developing single tiered and single threaded application, but Volta does not offer solutions to solve these timing problems.

Tier-splitting in Volta still introduces timing problems a developer might not be expecting. Insight in distributed computing is still required for programming a Volta application that is split into multiple tiers.

Q 1.5 What restrictions does Volta impose to languages?

C# is the current main language which Volta currently aims at. C# is also officially supported, but Volta implies a few restrictions. Answers on this question are mainly concentrated to this language.

C#

The first major constraint is that inheritance is not possible for any class if this is to be run on the server tier. Every server-side class has to inherit directly from the `Object` class. It is still possible to use, build and implement interface classes. It is also possible to create abstract classes, but since it is not possible to extend them, they do not seem very useful.

Running methods with more than one return statement resulted in unexpected behavior. Instead of returning on the first return statement, the program breaks out of the current block and continues running.

Parameter passing by reference keywords `out` and `ref` are not supported in Volta. Normally these keywords are used in a similar way as pointers are used in C when passing parameters to functions.

Visual Basic

During this research, Visual Basic has not been examined, but here is an overview from the Volta website on known Visual Basic issues:

- Late binding is not supported
- Services from `My` are not supported
- Not all conversion functions (for types) are supported
- Not all legacy code is supported

Other .NET languages

The *Base Class Library*, which is usable by every .NET language, is not fully supported. The precise (im)possibilities are not examined here and are more likely to be more useful when the Base Class Library is better supported.

Volta still imposes several limitations on the usage of C#. Inheritance is not fully supported, return statements are not handled as expected and out and ref do not work. Other .NET languages are also not fully supported and de Base Class Library poses significant restrictions for programming in other languages.

Q 1.6 Is it possible to use the same Volta tools for every supported language?

This question was abandoned. Volta mainly supports the development of C# and partly supports Visual Basic. Of course ordinary .NET connectivity between languages can be used to import functions in C# and Visual Basic as this is a feature that comes with the CIL assembler of the .NET framework (10). Testing whether Volta operates well would be more interesting from the moment that Volta fully supports tier-splitting for the total .NET Base Class Library.

Results of Main question 1

Volta enables tier-splitting in general, but there are still restrictions that are needed to be solved in order to present developers a fully automated tier-splitting compiler. The main restrictions are:

- Incomplete support for the Base Class Library
- Inheritance in C# is not fully supported
- Tier-splitting is limited to one server and one client (the browser) tier

Brownfield scenarios were not fully researched, but the expectation is that the deployment of Volta to existing systems will not pose many problems. Question 4 discusses the capabilities of Volta to cooperate with other technologies.

Volta does not solve problems introduced by distributed processes. Shared variables and timing problems are still the responsibility of the developer. This does imply that a developer needs at least a notion of distributed computing in order to safely implement a tier-split application.

19. Main question 2: What is the performance of Volta?

Developing Internet applications is not new: there are other technologies available. It is important to know if developing an application developed with Volta will perform significantly different. Another question related to performance is whether Volta enhances the performance of the developers. Since both these aspects will be discussed, Main question 2 will be split up in Main question 2a: Run-time performance and Main question 2b: Developer performance.

19.1. Main question 2a: Run-time performance

This section will describe the performance of Volta. This relates to response times of functions as well as resource usage. The sub questions discussed here are:

1. How many resources will a Volta application require to operate with acceptable response times?
2. What are the effects of tier-splitting on the performance of a Volta application?
3. What is the difference between running an operation in the browser at the client side compared to running it server-side?

Because the output of the Volta compiler is not optimized yet, these questions could not all be answered completely.

Q 2.1 How many resources will a Volta application require to operate with acceptable response times?

It is very difficult to give an unambiguous answer to this question. One thing that should be noted again is that the JavaScript output of the Volta compiler is not optimized yet (11). What optimizations a future release could bring and how this will affect the response time and resource usage is unknown.

For the current release however, Volta generated 2725 files for a simple *Hello World* webpage. This required 25.8 MB of hard-disk space on the server. Obviously this is significantly larger than an ordinary webpage of one HTML-file that takes up 386 bytes. Furthermore, the large number of JavaScript files makes a Volta page less responsive than a comparable webpage. Published at the project team's Web server, the Hello World webpage took around three seconds to load.

No further research has been done for this question because of its ambiguity and the lack of time. It is recommended to revisit this question when a final release for Volta is available.

Q 2.2 What are the effects of tier-splitting on the performance of a Volta application?

This question cannot be answered completely yet. As mentioned in Q 2.1, the output of the Volta compiler is not optimized yet.

It is uncertain whether an application built with the release discussed in this report will reflect the highest performance possible for Volta. It is recommended to research this issue again when a final release for Volta is available. No further research has been done for this question.

Q 2.3 What is the difference between running an operation in the browser at the client side compared to running it server-side?

For the current release, there is a difference in what operations can be run efficiently on the client side and what can be run efficiently on the server side. The difference varies for different tasks. Again, it should be noted that the current release of Volta is not optimized for performance yet.

An example Volta page has been created that shows the current difference for arithmetic operations. It calculates the n^{th} Fibonacci number recursively and returns this in a Paragraph on a Web page. The calculation is done in a class that can either run on the client side or server side. The program code is provided in appendix B. The *Rotunda end-to-end profiling* that comes with the Volta installation was used to find out the exact run times. Figure 6 and Figure 7 show the run times in milliseconds for various numbers of *n*. Every bar represents the average of five runs in that particular situation.

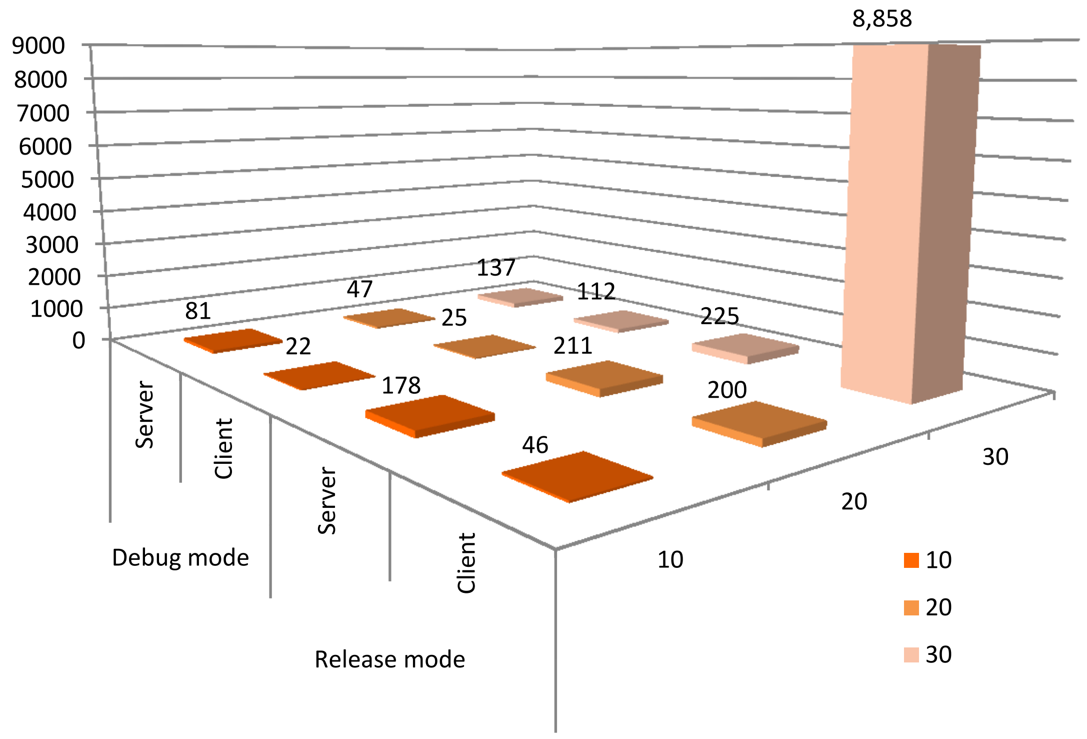


Figure 6. Response in milliseconds for calculation the 10th, 20th and 30th Fibonacci number

In Debug mode, the program is not compiled to actual JavaScript, so the differences between computing the value client-side rather than server-side should be marginal. For low values of *n*, this actually is not the case, but since the response time is so low, minor changes in the run-time environment might have influenced the results. For higher values of *n*, the differences become negligible.

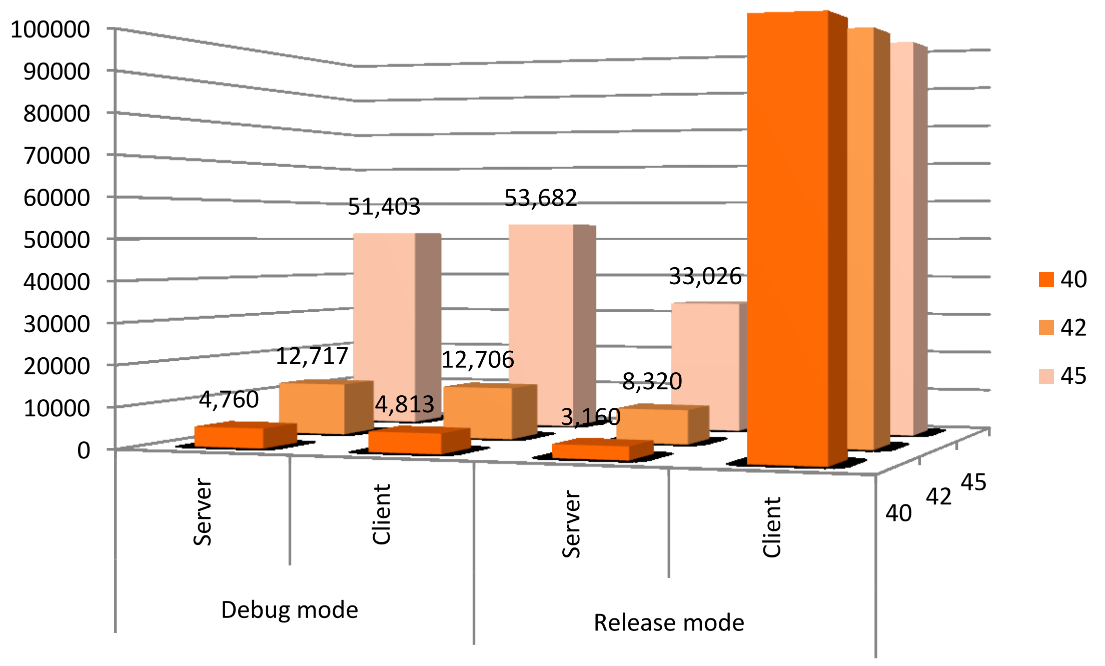


Figure 7. Response time in milliseconds for calculating the 40th, 42nd and 45th Fibonacci number

In Release mode however, the client code is converted to actual JavaScript. In this mode, run times significantly increase for higher values of n, when the computation runs client-side. For low values of *n*, the client outperforms the server. For *n* > 20 however, this transforms to an advantage for running server-side. For *n* > 40 when running client-side, the browser stopped responding within reasonable time: after ten minutes, the browser was still blocked. This behavior is represented by a value of 100.000 in the graph.

Also noticeable is the performance in Release mode of the server, compared to Debug mode. It is possible that the program code for Release mode is optimized to a certain extent, since it does not have to account for possible breakpoints or other specific functions needed for debugging. We have posed this question to the Volta developers, but have not yet received an answer.

The test points out the use of the ability to easily switch classes to different tiers for performance improvements. There is a significant difference in run times of arithmetic operations when run either on server-side or client-side. With the current release of Volta, the client side is not efficient enough for heavy calculations.

Results of Main question 2a

It is difficult to assess the actual performance of Volta. The code generated by Volta is not optimized for speed, which is also emphasized by the developers of Volta. Load times are higher than one would expect for a certain page due to the many JavaScript files. When a final release of Volta is available the performance can be analyzed completely. With the current release some operations perform better on a specific tier. This points out the usefulness for easy tier-splitting.

19.2. Main question 2b. Developer performance

Besides the performance of the actual Volta applications, it is interesting to research how Volta can simplify processes for developers. In this section, the following sub questions are discussed:

4. In which way does Volta support the development of feature rich applications?
5. How much does Volta contribute to ease the development of an Internet application?
6. How easy does a Volta application get tested in order to keep security and bug issues close to zero?
7. How does testing with a Volta application relate to ordinary Web development testing?

Q 2.4 In which way does Volta support the development of feature rich applications?

This question is discussed in Q 2.5, since both questions discuss the same subject.

Q 2.5 How much does Volta contribute to ease the development of an Internet application?

To be able to assess the convenience of developing with Volta, a number of criteria was inspected. These criteria are the working environment, since this can increase a developer's productivity; the teamwork abilities because software development primarily takes place in teams; and the convenience of tier-splitting since this is a specific feature of Volta.

Working environment and teamwork

The first two aspects, the working environment and the teamwork abilities, are easily answered. Volta applications are built in the Visual Studio 2008 IDE. Visual Studio enforces a structured method for setting up projects. Visual Studio's IntelliSense and the compiling presets of the Volta installer made the work of application development quite easy. Unfortunately, Volta does not provide documentation for its functions, as mentioned in section 17.2, so IntelliSense cannot be used for this. The TFS functionality makes it possible to share files with each other, keep track of different versions and lock files when necessary. The Volta Visual Studio plug-in also provides a toolbox specially for creating HTML pages.

Tier-splitting

Tier-splitting a Volta application can be done by inserting a class attribute. By placing the `[RunAtOrigin()]`[8] attribute above a class one defines that this class will be compiled CIL to be run server-side. Classes without this attribute are compiled to JavaScript and run client-side (i.e. in the browser).

Volta's method for tier-splitting is much easier for developers, since a developer does not have to worry about the communication code. It is difficult to assess what exactly happens when this goes wrong, but the Rotunda end-to-end profiler helps if this occurs.

Debug and Release modes

Debugging and releasing a Volta application introduces new problems. An application that works in Debug mode does not necessarily work in Release mode, which is already mentioned in section 17.2. For example: asynchronous applications can be built easily in Debug mode, but will fail in Release mode. With some editing, asynchronous processing can be implemented, but requires another review of the source code by

[8] Future releases plan to implement a general `[RunAt("TierName")]` attribute.

the developer. These differences do not make it very suitable to develop in Debug mode: too much code needs to be changed to effectively use this mode.

The Document Object Model

A notable feature is that the HTML library is mainly based on the Document Object Model. Although there is no official support to the DOM it seems like it works up until level 2. This does ease the development by people that are already well familiar with this widely used model. Firefox and Internet Explorer are officially both supported, but only Internet Explorer seemed to work at less simple application scenarios.

Volta does not make developing an Internet application much easier. This is primarily because of the lack of documentation and difference in compiling modes, which has already been mentioned in chapter 17. Documentation can easily be provided in a future release, but how the difference between Release and Debug mode will evolve in future versions is not clear. Convenient elements of Volta are the tier-splitting possibilities and the support for the DOM.

Q 2.6 How easily can a Volta application be tested for security and bugs?

On the Volta website, there are some remarks about debugging and testing a Volta application. In the FAQ section, the website states the following about the intended benefits to developers: *“Developers can use all the .NET languages, libraries, and tools they already know, including debuggers, profilers, test generators, refactoring, and code analysis tools. […] Moreover, Volta supports end-to-end profiling and testing in a manner that cannot be achieved with different programming models in the browser and server tiers.”*(11) Visual Studio can auto-generate a number of tests for a project.

Web tests

First of all, *Web tests* can be used for several Web related tests, for instance: timing a HTTP-request, or validating the contents of a response. For Volta, response times can be checked, but nothing regarding the actual content of the Web page. The tests validate the HTML code of the page. However, Volta uses JavaScript to build the page, which means the test will receive a nearly empty HTML page. There are some frameworks available for testing JavaScript functions, but the Volta-generated code is too complex to be able to create meaningful tests. This makes it very tedious, if not impossible to test with these frameworks. Automatic testing of Ajax enhanced Internet applications is currently a subject of research at Delft University of Technology (12).

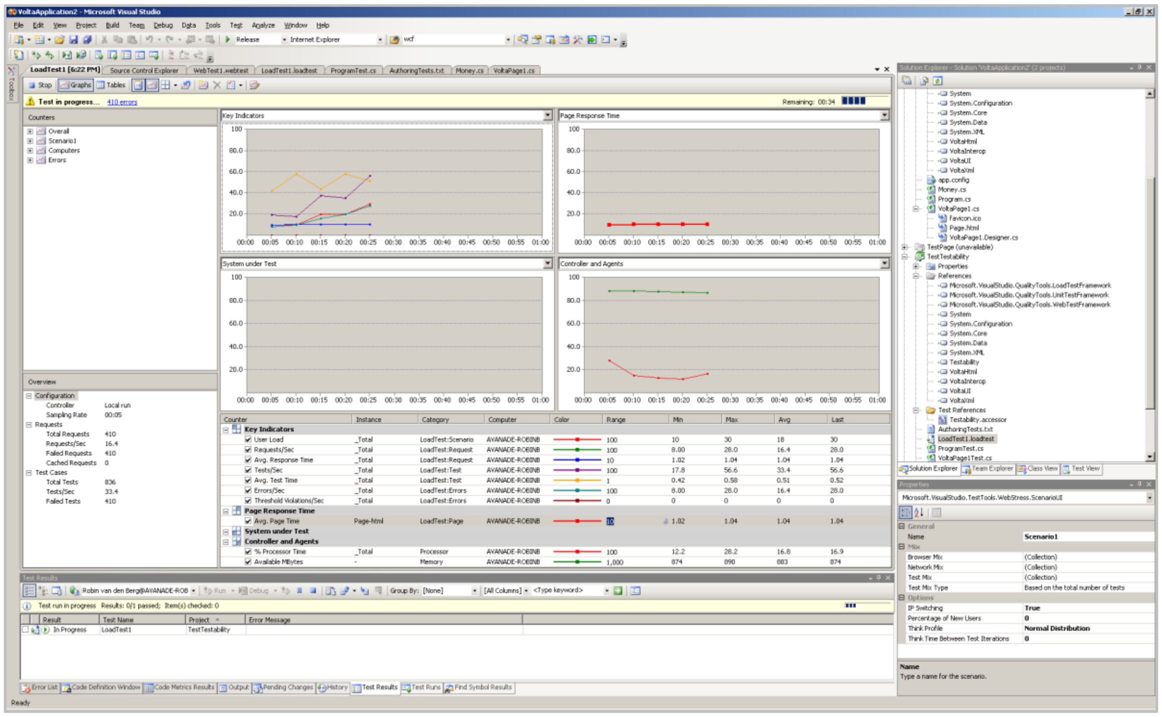
Unit tests

Unit tests can help significantly for finding bugs in the business logic. The framework Visual Studio provides for the tests is comparable with IDEs like Eclipse, and provides startup and teardown methods, assertion checking, result overviews and coverage reports.

Unit tests can be run on the classes of a Volta project, though it should be noted that the code will not run as actual JavaScript, but in the CLR as Intermediate Code. This means that the CIL to JavaScript compiler cannot be Unit tested. This is very inconvenient for the development of an application in the current release of Volta, because a number of differences between compiling in Debug mode and Release mode came to light. These are discussed in section 17.2.

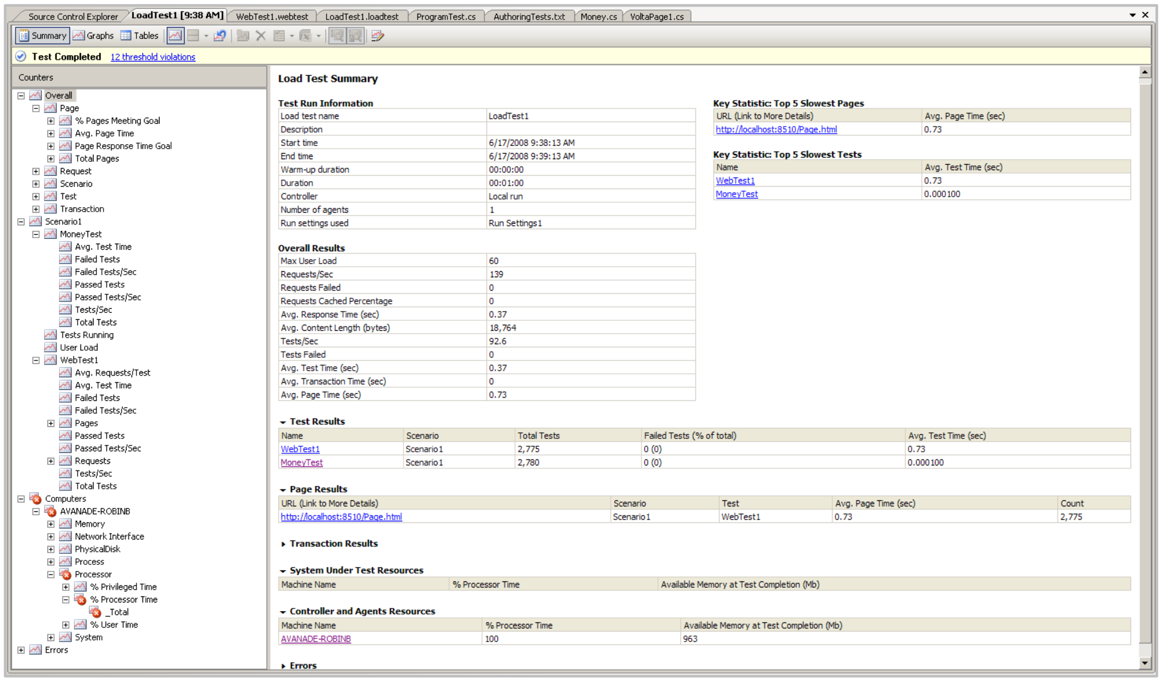
Load tests

Visual Studio can also generate load tests. With this, several values can be tested when a Volta page is under heavy load. Basically it entails that multiple other (unit) tests are performed simultaneously and in parallel. The same restrictions for the Web tests are valid: it cannot test the contents of the page generated by Volta. Figure 8 and Figure 9 offer an overview of a test screen in Visual Studio and a summary of the results, respectively.



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg	Last
Key Indicators									
☑ User Load	_Total	LoadTest:Scenario	AVANADE-ROBINB	🟥	100	10	30	18	30
☑ Requests/Sec	_Total	LoadTest:Request	AVANADE-ROBINB	🟩	100	8.00	28.0	16.4	28.0
☑ Avg. Response Time	_Total	LoadTest:Request	AVANADE-ROBINB	🟦	10	1.02	1.04	1.04	1.04
☑ Tests/Sec	_Total	LoadTest:Test	AVANADE-ROBINB	🟪	100	17.8	56.6	33.4	56.6
☑ Avg. Test Time	_Total	LoadTest:Test	AVANADE-ROBINB	🟧	1	0.42	0.58	0.51	0.52
☑ Errors/Sec	_Total	LoadTest:Errors	AVANADE-ROBINB	🟦	100	8.00	28.0	16.4	28.0
☑ Threshold Violations/Sec	_Total	LoadTest:Errors	AVANADE-ROBINB	🟥	0	0	0	0	0
Page Response Time									
☑ Avg. Page Time	Page-html	LoadTest:Page	AVANADE-ROBINB	🟥	10	1.02	1.04	1.04	1.04
System under Test									
Controller and Agents									
☑ % Processor Time	_Total	Processor	AVANADE-ROBINB	🟥	100	12.2	28.2	16.8	16.9
☑ Available MBytes	-	Memory	AVANADE-ROBINB	🟩	1,000	874	890	883	874

Figure 8 Screenshot of a Load test in Visual Studio



Test Completed 12 threshold violations

Load Test Summary

Test Run Information

Load test name	LoadTest1
Description	
Start time	6/17/2008 9:38:13 AM
End time	6/17/2008 9:39:13 AM
Warm-up duration	00:00:00
Duration	00:01:00
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Key Statistic: Top 5 Slowest Pages

URL (Link to More Details)	Avg. Page Time (sec)
http://localhost:8510/Page.html	0.73

Key Statistic: Top 5 Slowest Tests

Name	Avg. Test Time (sec)
WebTest1	0.73
MoneyTest	0.000100

Overall Results

Max User Load	60
Requests/Sec	139
Requests Failed	0
Requests Cached Percentage	0
Avg. Response Time (sec)	0.37
Avg. Content Length (bytes)	18,764
Tests/Sec	92.6
Tests Failed	0
Avg. Test Time (sec)	0.37
Avg. Transaction Time (sec)	0
Avg. Page Time (sec)	0.73

Test Results

Name	Scenario	Total Tests	Failed Tests (% of total)	Avg. Test Time (sec)
WebTest1	Scenario1	2,775	0 (0)	0.73
MoneyTest	Scenario1	2,780	0 (0)	0.000100

Page Results

URL (Link to More Details)	Scenario	Test	Avg. Page Time (sec)	Count
http://localhost:8510/Page.html	Scenario1	WebTest1	0.73	2,775

Transaction Results

System Under Test Resources

Machine Name	% Processor Time	Available Memory at Test Completion (Mb)

Controller and Agents Resources

Machine Name	% Processor Time	Available Memory at Test Completion (Mb)
AVANADE-ROBINB	100	963

Errors

Figure 9 Overview screen of results of a load test

Testing Volta applications resembles testing other applications in most areas. Volta enables using the test facilities of Visual Studio. Fully automated testing of a complete Volta application is not possible yet. When the testing of generated JavaScript can be automated, testing a Volta application will not differ significantly from testing other applications.

Q 2.7 How does testing with a Volta application relate to ordinary Web development testing?

Since a Web application can be developed in C# with Volta, a developer can use the unit test and load test functionality of Visual Studio, as described in the section above. This only covers the pure C# code, but extensive unit testing frameworks are still not commonly available for other technologies. For the actual webpage generated, Web tests cannot simulate all the behavior, but as already mentioned above, Ajax testing is still a subject for research regardless of the used technology (12). This means being able to develop in Visual Studio with Volta and C# is an advantage over other technologies that do not have unit testing frameworks.

Results of Main question 2b

There is a number of things to be said about the developer performance of Volta. Besides this, the lack of documentation makes it very difficult to create a real project. During the development of GReenGRasp, we continuously had to change the implementation plan because some methods or classes were not supported, or generated incomprehensible JavaScript-errors. The short recipes provided on the Volta project website do not provide enough information for a realistically sized project.

Testing a Volta application resembles testing a regular application in Visual Studio, which is an advantage when compared to other Web development technologies. The primary concern here is testing the actual results the JavaScript produces in a browser. This is a general issue with Ajax applications though, which is already subject for current research.

20. Main question 3: How does Volta contribute to secure development?

It is still difficult to realize a fully secure application in most projects. For a new technology, this can play an important part in its acceptance. In this chapter we research how security is dealt with in Volta. Also the support provided to the developer eases acceptance.

Q 3.1 Does Volta make it easier to construct error-free communication code?

When using Volta, the programmer does not have to manually set up the communication between tiers. The Volta compiler automatically generates the required code between the `[RunAtOrigin()]` classes and the client-side classes. The programmer has no influence on this generated code: the communication between tiers is more or less a black box. An advantage of the code generation is that the programmer cannot introduce bugs in the communication. Debugging generated code is far from easy though.

Volta does not avoid assumptions of distributed computing (13) and thus does not preserve a programmer from having to worry about these issues. These issues are discussed more thoroughly in Q 1.4 and Q 1.5.

Q 3.2 Does the JavaScript code generated by Volta contain obvious leaks?

The automatically generated JavaScript code is not designed to be human readable. Variables like function names like `aTD()` and `tSDF()` are the best references the JavaScript shows. A simple Volta application exists of around 15.000 different JavaScript files divided over more than 50 folders. This shows that reverse engineering will not be an easy task.

Further research has not been done, because the generated JavaScript is still under development in order to significantly improve performance.

Leaks in the generated code of Volta are not obvious due to the complex and unreadable structure the generated JavaScript code. Further research should be done when JavaScript generation has been optimized in a newer release.

Q 3.3 Can communication between tiers be established securely using security protocols such as SSL or TLS to prevent transferred data interception and manipulation?

Volta does not support any encryption methods when sending sensitive information across the network. Generating MD5- or SHA1-hashes is not possible. Of course, it is possible to implement a custom encryption in a Volta application. In theory however, this could makes it possible to find out the exact encryption method if this is tier-split to the client-side, although reading the auto-generated JavaScript is nearly impossible. In practice, it is much easier to implement a secure connection created by the Web server. This way, the connection is always encrypted and one does not have to consider encrypting variables sent across tiers.

A Volta application runs in a Microsoft Internet Information Server (IIS). IIS has a build-in option to serve SSL secured websites. The GReenGRasp showcase is fully tested with the SSL connection and posed no problems. Connections were successfully encrypted here and the application did not show unexpected errors caused by the SSL security.

```
{__type:"Microsoft.LiveLabs.Volta.MultiTier.CallInfo",Type:"Account",Met
hod:"4",Instance:{__type:"GReenGRasp.DataObjects.Account",id:"15328412-
81c1-432f-96b5-177bfef9b0dc"},Tag:null,
Parameters:[{__type:"System.String",value:"Andre"}]}

{__type:"Microsoft.LiveLabs.Volta.MultiTier.CallInfo",Type:"Account",Met
hod:"8",Instance:{__type:"GReenGRasp.DataObjects.Account",id:"c5da4e62-
bf96-47a6-aeed-26cde0aaad81"},Tag:null,
Parameters:[{__type:"System.String",value:"ditwachtwoord"}]}
```

Code sample 4. Part of TCP/IP packet caught by Wireshark

For the GReenGRasp project, the network packets were monitored when using an early version of the login-control. The username and password could clearly be identified, as Code sample 4 shows. An SSL connection makes this communication unreadable (for this reason no code is provided).

In general, encryption functionality is not supported by Volta. It is possible to encrypt messages manually by the implementation of a custom encryption method. A better encryption solution is to use the IIS SSL certificate functionality to encrypt all the communication between the browser and the server.

Q 3.4 Is it possible to implement anti spoofing mechanisms and does Volta facilitate it?

Spoofing associated with websites is represented in different forms. Man-in-the-middle and phishing (URL spoofing) attacks are examples of Web spoofing problems that a developer is faced with. Spoofing is in essence the fooling of a user in order to extract his or her private information. The only security measure a website can offer to these attacks is to offer a login account for a user which saves sensitive information on the website behind a password and taking care of a secure connection. Spoofing by giving a different URL (i.e.: www.greengresp.net instead www.greengrasp.net) for example cannot be prevented, even by certificates.

Volta provides a built-in login control, but creating such a control manually is not very difficult. In the GReenGRasp showcase a control has been implemented. The best solution here was to encrypt the connection through an SSL secured connection as described in Q 3.3.

Volta does not present any new anti-spoofing mechanisms. Volta does not make it harder to implement anti-spoofing mechanism and known techniques are not hard to implement. An SSL connection is also easily set up.

Q 3.5 Is it possible to produce web applications/pages invulnerable to code injection and does Volta help prevent this?

With Volta, the general way to process data from forms is to fetch the values with JavaScript. It is possible to check the input for code injection on the client side, as well as the server side. The programmer should pay attention though, since by default the classes run on the client side and client-side verification can be bypassed by a user. This can be done by editing the JavaScript. This will not be easy since the code is Volta-generated, as mentioned in Q 2.1. Tier-splitting the verification to the server side makes this unreachable for a user and thereby is more secure.

The C# .NET library provides functionality for processing user-provided parameters securely if they are used in database queries. This functionality relies on parameter binding. This processing functionality comes in handy to filter out code injections in queries.

There are several functions a programmer can use for validating user input. Functions like *contains* and *replace* are standard string operations of the C# library. Unfortunately, there is no function to automatically convert a user-provided string into a secure string where selected characters are filtered out. This could be a useful addition for Volta, although it would have to contain a construction for automatically execute it on the server-tier. As mentioned in Q 2.1 and Q 2.3 the verification should ideally be performed there. The JavaScript on the client-side could be altered by the user, making it possible to circumvent the measures against code-injection. The server tier does not include this vulnerability.

Volta's generated JavaScript is harder to grasp compared with human generated JavaScript code, but it is still not impossible to bypass. Some C# .NET facilities do make it easier to filter out code injection, but not everything is fully supported. In general Volta supports the development of secure applications, but this is still the responsibility of the developer.

Q 3.6 How does Volta contribute to security assurance? And is it easier to assure security with Volta than with common web development tools?

The possibilities of testing a Volta application are discussed in section 19, Q 2.6. Volta makes it possible to use C#'s functional and testing possibilities in Visual Studio to test Volta applications build in C#.

When it comes to security, a programmer using Volta has to think a bit more about security in comparison with other applications. Since tier-splitting in Volta is very easy, classes can be created without needing an explicit notion of on which tier to run it (as this is decided after implementation). In languages like PHP or ASP.NET, methods are always run on the server and it is necessary to explicitly create JavaScript functions in order to use browser side programmability. One important notion that has to be included into this decision is the fact that everything that runs on the client (in JavaScript) can be altered by the user. This sometimes means that additional functionality is required for security reasons. When using Volta, a programmer will be less aware of this since classes run on the *client* by default, and a programmer might forget think about the implementation of the required security fail-safes when splitting up the application.

The discussion about the notion of tier security issues is in some way the same as the discussion about the level of abstraction a programming language should reach. In Volta a programmer should be aware of the implications of easy tier-splitting and be cautious of security issues raised by targeted tiers.

Secure development is not simplified by Volta. A developer even has to be aware of the security issues that are implicated by the targeted tier.

Results of Main question 3

Developing security mechanism in Volta is not made easier, but also not harder. Volta does not deliver anti-spoofing mechanisms to ease security issues.

Using an SSL connection would be a wise decision to provide a more secure Volta application. The combination of the badly readable JavaScript and the encryption creates poses new problems for possible malicious usage. This should make a Volta application more secure than ordinary applications protected with SSL.

A drawback of the abstraction of tiers during the development is that a developer is not aware of the security issues of the targeted tier. These security issues are also not solved by Volta.

21. Main question 4: How does Volta cooperate with other technologies?

A programmer spends most of his time adjusting existing programs, fixing and extending legacy code. Also, many Web applications rely on different technologies, which have to communicate with each other according to standards. It is important to know how a new technology can form an extension to already present systems. This allows switching to this technology in a more robust and gradual way. This chapter discusses the ways in which Volta can cooperate with other technologies.

Q 4.1 Which standards does Volta apply to already?

The answer to this question is provided in Q 4.4 because of the relation with the subject discussed in that sub question.

Q 4.2 Which standards will be implemented in the near future?

Since there is no roadmap for Volta (11), this sub question cannot be answered and was therefore abandoned.

Q 4.3 Would there be any important standards that Volta is not going to support?

This sub question was abandoned for the same reason as Q 4.2.

Q 4.4 In what ways can Volta be used by other technologies?

Other technologies can benefit from Volta if they can successfully communicate with Volta. This enables Volta to extend current technologies, which will increase its usefulness. Three general technologies are discussed below: how Volta can be used by preprocessing Web technologies or by browser-based virtual machines.

Preprocessing Web technologies

A Web application based on preprocessed code like ASP.NET is often supported by JavaScript code in order to enrich the user experience of an application. If one root HTML source file is used, a connection with a preprocessed Web application can be made. If more are used, which is usually the case for realistic Web applications, illegal `GetElementByID("targetedHtmlElementID")` calls cause errors. This is something to be tackled by the Volta developers. However, if it is possible for a Web application, integrating the ASP.NET and Volta part in one solution with two projects that refer to the same HTML source file can currently already make development of a rich ASP.NET application easier.

Browser based virtual machines

Web applications running on browser-based virtual machines like Microsoft Silverlight can be supported by Volta. The showcase provides an example for this: the Silverlight code uses Volta code for the business logic.

As long as such a Web application supports the communication through JavaScript it is possible to communicate with Volta. By using Volta with these technologies by passing events through JavaScript it is possible to create applications that have a client-side and server-side application tier.

Other technologies, both preprocessing and running in a browser-side virtual machine, can be extended with Volta. However, the ways to achieve this extension are not easy and limit functionality, which makes it difficult to assess the actual usefulness. Simplification of these methods could be possible and would make Volta a useful extension for other technologies.

Q 4.5 How easily can Volta be used with another technology?

This question was abandoned due to lack of time.

Q 4.6 Which technologies of Microsoft are supported?

This question is discussed integrally with Q 4.8, Q 4.12 and Q 4.13 in the new question NQ 4.1.

Q 4.7 What is the overlap in these technologies?

This question was abandoned due to lack of time.

Q 4.8 How does the integration work?

This question is discussed integrally with Q 4.6, Q 4.12 and Q 4.13 in the new question NQ 4.1.

Q 4.9 Does Volta provide a way to use Web services?

It is possible to make use of Web services both on the client and the server side. Volta can create and send XmlHttpRequests and receive responses. The next sub question elaborates on how this can be achieved.

Q 4.10 How easily can we use these Web services?

Using the XmlHttpRequest-object and the example on the website of Volta, Web services can be used very easily (14). The XmlHttpRequest-object preserves the semantics as typically used in non-Volta Ajax-style applications. Therefore, developers already familiar with it do not need to learn anything new. An example is given in Code sample 5.

The Visual Studio IDE also supports adding *service references* to a project, which enables access to one or more WCF services. For Volta though, this does not completely work. The IDE consistently crashed when this was tried for the Smart Metering Web service, and others have encountered similar problems (15). Signing the assembly of the reference might solve this, but this is not always possible: a Web service is usually independently created by a third party. This was also the case with the Smart Metering Web service for the GReenGRasp website.

Alternatively, it is possible to use WCF services in program code. First, a DeviceDataService class and a corresponding configuration file have to be generated using the ServiceModel Metadata Utility Tool shipped with the Windows SDK. The functions of the specific Web service are automatically created within this class. A side effect is that it is not possible to use the configuration file. In that case the variables of the configuration file have to be defined in the program code. A bigger problem is the fact that parts of the DeviceDataService class are "not supported" by the Volta Compiler, which means it is not able to run client-side. Changing this to the server-side tier cannot be done by simply applying the `[RunAtOrigin()]` attribute. This is because the DeviceDataService class inherits from a WCF library class and inheritance is not supported server-side, as explained in Q 1.2. The workaround explained in section 13.2 can be used to solve this problem.

```
public String getWebService()
{
   String result = "";

   // the address of a web service
   String uri = "http://ajaxify.com/run/xmlHtttpRequestCall/
                     sumGet.phtml?figure1=5&figure2=10";

   // create a new request
   XMLHttpRequest xhr = new XMLHttpRequest();
   xhr.Open("get", uri, false);
   xhr.Send();

   // process the response
   if (xhr.Status == 200)
   {
      result = xhr.ResponseText;
   }
   else
   {
      result = "error";
   }

   return result;
}
```

Code sample 5. Example method that uses a Web service

The conclusion can be drawn that simple Web services can be used relatively easily. It is more difficult however to use WCF Services. Building support for the functions that the ServiceModel Metadata Utility Tool generates would simplify the use of WCF. Allowing classes to inherit from System.Object, or removing the requirement of inheriting from this class in total would provide a welcome solution.

Q 4.11 In which way can Volta itself provide Web service(s)?

Creating Web services is not one of the intended purposes of Volta. It is possible, but not recommended to do so.

The main difficulty is that Volta cannot access the variables that are sent with the request for the Web page. This means it can only access variables by trying to parse parts of its URI, in case the request is sent with a GET method.

Another limitation is that the only way Volta can respond directly is with an XmlHttpRequest, a database command or its HTML page that is built by client-side JavaScript. These cannot be sent directly back to the caller of the Web service. In this case only a one-way request is possible. However, this does not follow the principles of REST which are important factors of the Internet (16). REST states that the GET method should only be used for information retrieval which is something a Volta application cannot accomplish.

The example of Code sample 6 uses the `Search` property of the location of the document and parses this string to get the necessary variables. It can then do whatever the Web service is required to do. This was the only method found to provide a Web service in Volta.

```
public void webService()
{
   if (Document.Location.Search.Length > 0)
   {
      // uri = ...page.html?id=value1&newvalue=value2
      // Search retrieves the substring starting with '?'
      String GETvars = Document.Location.Search.Substring(1,
         Document.Location.Search.Length - 1);

      String[] varsarray      = GETvars.Split(new char[] { '&' });
      String[] id             = varsarray[0].Split(new char[] { '=' });
      String[] newvalue       = varsarray[1].Split(new char[] { '=' });
      int value1         = (int)id[1];
      String value2      = newvalue[1];

      // followup actions
   }
}
```

Code sample 6. A Web service method in Volta

Concludingly, it can be stated that a simple Web service can be provided by Volta, but this is too tedious to use. Providing Web services is not a goal of Volta and is not recommended.

Q 4.12 What are the possibilities to cooperate with technologies of competitors?

This question is discussed integrally with Q 4.6, Q 4.8 and Q 4.13 in the new question NQ 4.1.

Q 4.13 What are the possibilities to cooperate with technologies of partners?

This question is discussed integrally with Q 4.6, Q 4.8 and Q 4.12 in the new question NQ 4.1.

NQ 4.1 With which technologies can Volta cooperate and how does this work?

For this issue we have examined five Microsoft technologies: SQL Server 2005, LINQ, WCF, Silverlight and ASP.NET (in the order mentioned). Although Volta does not natively support four of the five products, it is capable of working with these products. A short description of the findings for each technology is provided below.

MS SQL Server

It is easy to implement a database connection to MS SQL Server. Volta supports the ordinary database connection methods that are normally used in other .NET applications. In order words: code used in Volta code be copied and used in any other .NET application and vice versa.

LINQ

LINQ was the second technology tried. Initially it worked in Debug mode, but it did not compile in Release mode. The reason for this is that Volta does not support inheritance on server-side classes as mentioned in section 18. Just before writing this report a workaround was came across that enabled the usage of WCF on the server tier. The same technique could be applied to LINQ on the server tier (see section 13.2).

Windows Communication Foundation

Integrating the Windows Communication Foundation presents the same challenge as LINQ does in Volta application development. For the WCF it is necessary to use .NET libraries that are not natively supported by Volta, but can also be implemented solely on the server tier (see section 13.2)

Silverlight

The communication of Silverlight with the Web server is based on the DOM. Because Volta also supports the DOM, communication with Silverlight is not very different from other JavaScript event communication. Using `Import` and `Export` directives of JavaScript events in C# made it possible to integrate Silverlight with the showcase.

Using Silverlight functions directly in a Volta application can be a promising possibility for the future. The advantage of Silverlight over JavaScript is that it *does* support multithreading and compiles to the same CIL code like every other .NET language does. Providing Volta with built-in support for Silverlight should not be hard to achieve in comparison with the DOM. It would only have to compile to CIL and XAML to have client-side code run in the browser. However, the difference in .NET frameworks used could complicate this task: Volta uses version 3.5 while Silverlight still uses 2.0.

ASP.NET

Using ASP.NET to support a Volta application can be done by taking a single HTML file (in this case a *.aspx file) as base for both applications. A developer does have to publish to a Web server twice (first the Volta application followed by the ASP.NET application) in order to properly compile the application. This means that Volta and ASP.NET have separate machine code and share their presentation layer. Combining the full .NET framework on the server side with ASP.NET and the advantages of tier-splitting and JavaScript generation from Volta can provide a powerful combination. To implement user management for the GReenGRasp showcase, ASP.NET based code was used to communicate with the Windows Live authentication Web service (see sections 13.3 and 13.4)

Although Volta does not natively support most of the tried Microsoft products, it is capable of working with these related technologies. SQL Server 2005, LINQ, WCF, Silverlight and ASP.NET can be used with Volta, but required workarounds that could and should be simplified.

Results of Main question 4

Using Volta in cooperation with other technologies is possible via importing and exporting JavaScript functions. For most related technologies, additional workarounds were required to accomplish cooperation. In a future version, most of these workarounds could be simplified, making Volta a more useful extension for these technologies. The cooperation with Web services is something that this release already simplifies, although WCF also required a workaround. Providing Web services with Volta is not recommended.

22. Main question 5: How does Volta perform in comparison with resembling and competing technologies?

Since Web development received a new impulse by the Web 2.0 movement several toolkits and technologies have been introduced. Most of these tools aim to offer developers easier ways of creating rich collaborating Internet applications which target SaaS products. How Volta performs in comparison with resembling technologies determines largely whether it will be adopted by companies. This chapter aims at the comparison of Volta with other technologies.

Q 5.1 Which technologies have similar if not comparable techniques to Volta and how do they ease the pain of SaaS driven development?

Google Web Toolkit

GWT aims at the same goal of easing the development of web applications. It does so by providing a compiler that compiles Java code to JavaScript. This also means that developers are able to use existing Java tools. The important differences of GWT compared with Volta are:

1. GWT does not aim at tier-splitting;
2. GWT does not aim to abstract the browser from the developer.

Script#

Script# is similar to GWT, but aims at the .NET Framework instead of the Java Open Source community. It does not aim at tier-splitting and the abstraction of the browser layer.

GWT and Script# both aim at easing development by compiling existing framework code to JavaScript and do not support tier-splitting facilities.

Q 5.2 How does Volta perform compared with other currently used techniques that strive to facilitate SaaS driven development?

First of all, tier-splitting just before compile time introduces a new concept. Volta proved the possibility of tier-splitting from Web server to Web browser with the help of the DOM. Not everything is already possible in the browser tier like multithreading and direct calls from server to browser, but these browser challenges are also not overcome by other technologies either.

One thing that is already on the Volta teams' agenda is the optimization of the JavaScript code. We agree with the fact that this should be optimized. As an example, a simple chatter program that took around two to three seconds to load its JavaScript code in order to start. In comparison with the performance of applications build with the Google Widget Toolkit like Google Docs it seems that this Volta CTP is way behind. Script# currently offers more support to the .NET framework (reflection, for example).

In short it is too early to say whether Volta provides better or worse results in comparison with currently launched software. Currently Volta is not capable to ease development more than GWT and Script# are capable of.

Q 5.3 What exact goals do these other technologies have and how does Volta perform in relation to these goals?

The goals of GWT and Script# are to deliver a compiler from one language to JavaScript (Java and CIL, respectively). Both do not target tier-splitting.

In Volta not everything works as expected and not all library functions are fully supported. Missing out on inheritance on the server side tier is also a big limitation.

Volta enables easier development in area of tier-splitting, because communication code is not an issue anymore. The only thing the developer should be very aware of possible security issues induced by a target tier. Still, the developer is able to concentrate on this instead of creating the communication code itself.

Comparing Volta with GWT and Script# is not possible one-on-one. GWT and Script# are more mature and support more functionality. Volta gives one great advantage of tier-splitting, but this is also not totally finished. For now GWT and Script# seem to be the better options for Web application developers.

Q 5.4 What are the practical results of other technologies in comparison with what Volta strives to achieve?

The most obvious candidates for this question are GWT and Script#, but due to a shortage of time it was not possible to examine these technologies. It would still be very interesting what their results are in a situation where Volta is in a more mature phase of its development.

Q 5.5 Does Volta simplify multiple tier development scenarios?

Easy development of multi-tiered applications is where Volta is able to show its big advantage. The power of Volta lies in its ability to split single-tier CIL code to multi-tier CIL code. The domain of Web development has been chosen to demonstrate this, but it has not been implemented completely enough to develop Web applications easier than established technologies. On the Volta website, the developers announce future possibilities for Volta: *"Further, execution contexts such as XNA, WinForms, DHTML, SVG, VML, even SQL are realistic client targets (tiers) for Volta."* (8)

Currently Volta does not fully simplify development of multiple tier applications. It is simple not mature enough to easily develop an application. In the case that other technologies and the .NET framework would be fully supported the situation would probably be different as Volta would be fully able to take care of the communication code.

Results of Main question 5

It is difficult to fully assess the performance of Volta in relation to other technologies, since Volta is in a very early stage of development. Volta is at a CTP stage, which is meant to demonstrate the basic idea. Comparable technologies are a lot further in the development cycle. Obviously, other technologies provide better support, documentation and performance. Also, they offered more complete solutions for security. On the other hand, the tier-splitting capabilities of Volta are not available in any other technology: this is Volta's unique selling point. Currently, other technologies for building Web applications perform better, but if Volta is further developed, it might catch up and eventually surpass them.

23. Research conclusions and recommendations

At the time of this report, Volta was at an early stage of development. To be able to give a good perspective of the value of this technology, an overview is provided of what Volta's current state is, what will be possible in the future and what issues Volta will not be able to solve.

In section 23.1, the point of view of the project team will be given, along with a number of recommendations for the future development of Volta.

What is Volta currently?

Volta is a technology that provides the ability to create a basic Web application that can be split into two tiers. There is a number of limitations that make the current release not fit for commercial use.

Volta can be used in cooperation with other technologies, but this often requires workarounds that are tedious to implement.

By using the Visual Studio it brings Volta the advantage of the available facilities for testing and collaboration.

Implementing security mechanisms in Volta is not easier or harder than in comparable Web development technologies.

What can Volta be in the near future?

Volta has several problems that can be solved in a future release. In order to overcome the restrictions that Volta currently imposes, it is possible to implement the following:

- More support for the Base Class Library;
- Inheritance for server-side classes;
- Extension of tier-splitting to more than two tiers

More documentation should be provided to be able to use Volta for a realistically sized project.

The performance of JavaScript can be improved in order to offer competing response times for Web applications.

In a future version cooperation with other .NET technologies can be simplified. This will allow Volta to be a complete Web application development tool.

Testing and debugging the generated JavaScript is currently very difficult. Improving the simulation that the Debug mode provides can solve this in the future.

What problems does Volta not solve

Volta does not solve problems introduced by distributing processes across multiple tiers. A developer still needs to know about the possible problems of distributed computing.

Another drawback of the abstraction of tiers during the development is that a developer might not be aware of the security issues of the targeted tier. These security issues are also not solved by Volta.

23.1. Recommendations

Volta is an interesting technology which promises to offer convenient tools for Web development. However, our research showed several unresolved issues in the current Community Technology Preview. Because of these issues we conclude that Volta is currently not fit for actual Web development projects. We recommend the development team focuses on the following issues in order to change this.

1. ***Provide more documentation.*** We continuously ran into problems that could have been prevented if the limitations and possibilities are better documented. Through trial and error certain limitations and workarounds were found for letting Volta cooperate with other technologies.
2. ***Equalize the behavior of Volta code in Debug and Release mode.*** It is very difficult to debug a program since the Debug mode differs from the Release mode. A developer spends too much time on debugging the actual application.
3. ***Decrease the number of restrictions on classes.*** This limitations make Web development difficult and obstruct the programmer from really being able to delay the choice of tier-splitting classes.
4. ***Increase cooperation with other technologies.*** Making Volta pages cooperate with these technologies required workarounds that could be simplified, while the technologies can offer very useful extensions to Volta and vice versa.
5. ***Optimize for performance.*** Currently a website created with Volta performs worse comparing to websites built with other technologies. This decreases the user experience of Volta websites.
6. ***Increase security functionality.*** Although it is possible to use encryption, other standard security functions have to be implemented by the developer. This costs time and brings the additional risk that JavaScript functions can be changed by the user.

Disregarding these issues, Volta can offer an easy way for tier-splitting an application:

- .NET languages can be used for developing Web applications.
- Developers do not have to worry about communication code of multi-tier applications.
- The facilities of the Visual Studio IDE can be used for testing, debugging and cooperating with other technologies.
- The end-to-end profiler is a useful tool for analyzing and optimizing a tier-split program.

However, developers should still be aware of the consequences of where certain program code is executed. In this sense, tier-splitting will always be difficult, but Volta speeds up the process following this decision.

All in all, the community technology preview of Volta looks promising. It is currently not fit for developing actual Web applications, but if the developers continue working on Volta and address the issues, Volta could prove to be a useful toolkit for Web development.

24. Conclusions

This report describes our project, the showcase and the research done on the Microsoft Web developing framework Volta.

The current version of Volta is a Community Technology Preview. Developing with an early release of a technology can result in unexpected situations. To account for these situations, sufficient time should be reserved in the planning for set-backs and familiarization with the technology.

Combining different requirements from multiple principals requires special attention.

The initial planning for this project was too optimistic. This optimism reflected our idea of Volta, acquired during the pre-research. However, this idea was based on the aims of Volta which is different from its current state. The roles and cooperation had a positive effect on the progress of this project.

The showcase does not demonstrate what can easily be achieved with Volta; instead it is an application that explores the boundaries of Volta. The experiences gained during the implementation of this showcase can contribute to the improvement of Volta.

The research showed that Volta is an interesting technology that promises convenient tools for Web development. However, several unresolved issues in the current release make Volta not fit for actual Web development projects. It should also be mentioned that Volta does not solve all problems for distributed computing. A number of recommendations for the development team have been provided which can improve the usefulness of Volta:

1. *Provide more documentation.*
2. *Equalize the behavior of Volta code in Debug and Release mode.*
3. *Decrease the number of restrictions on classes*
4. *Increase cooperation with other technologies.*
5. *Optimize performance.*
6. *Increase security functionality*

Lessons learned

Looking back on this project there are a few important lessons learned:

- Special care needs to be taken when using experimental and conceptual software. Take about twice the development time in order to keep a realistic planning when developing an application.
- A demonstration of an application needs preparation. Time and resources need to be reserved for this.
- Research questions tend to get too broad. Particular aspects should be researched and pointed out in the total picture. Optionally have some extra question in case there is time left, but try to focus.
- Beware of badly documented or undocumented software, especially when the software is experimental. The combination of no documentation and bad error messages often turn out in time consuming trial and error situations.
- Do not use multiple beta-, alpha- or other prerelease software simultaneously. This only complicates things even more.

Glossary

ASP.NET	A web application framework developed and marketed by Microsoft used by programmers to build dynamic Websites, Web applications and Web services. This is mainly based on preprocessed code generating Code-behind and DHTML.
Auto completion	Auto completion in a program predicts a word or phrase that the user wants to type in without the user actually typing it completely.
Base Class Library	The library that is available for every programming language in the .NET Framework
CIL	Common Intermediate Language is a low-level programming language resembling assembly to which all .NET languages are compiled. Also known as MSIL.
CMS	A content management system (CMS) is computer software used to create, edit, manage, and publish content in a consistently organized fashion
Code-behind	A model used in ASP.NET where dynamic code is separated from the static presentation layer. Code-behind typically takes care of data connections and event handling.
CTP	A Community Technology Preview is a pre-release version of software intended as a proof-of-concept for the community.
DHTML	Stands for Dynamic HTML collective term for the combination of technologies that enable dynamic web pages. A common website uses a markup language, a presentation definition language, a scripting language and the DOM.
DOM	The Document Object Model is a platform- and language-independent standard object model for representing HTML or XML and related formats.
End-to-end profiler	A tool that tracks the sequence of an application and allows analysis of messages between tiers. See Rotunda.
Flash	This Adobe product enables a virtual machine in a browser. Flash is mainly used to animations and interactive web applications.
Fluid	A webpage is fluid if all components resize proportionally.
IDE	An Integrated Development Environment provides comprehensive facilities for the development of software applications.
IntelliSense	A Microsoft's implementation of auto completion, best known for its use in the Microsoft Visual Studio integrated development environment. In addition to completing the symbol names the programmer is typing, IntelliSense serves as documentation and disambiguation for variable names, functions and methods using metadata-based reflection.
JSP	Java Server Pages is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request.

LINQ	Language Integrated Query (LINQ, pronounced "link") is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages using syntax reminiscent of SQL.
MS SQL Server	Microsoft SQL Server is a relational database management system (RDBMS) produced by Microsoft.
MSDN	The Microsoft Developer Network (MSDN) is the portion of Microsoft responsible for managing the firm's relationship with developers.
`out` (C#)	The `out` keyword causes arguments to be passed by reference. This is similar to the `ref` keyword, except that `ref` requires that the variable be initialized before being passed.
PHP	PHP: Hypertext Preprocessor; a computer scripting language, originally designed for producing dynamic web pages. PHP is solely executed server-side.
project (Visual Studio)	A configuration facility used by Visual Studio. Among these configurations it takes care of compilation directives, file management and the dependence of certain files on other files.
`ref`(C#)	The `ref` keyword causes arguments to be passed by reference. The effect is that any changes to the parameter in the method will be reflected in that variable when control passes back to the calling method.
REST	Representational State Transfer is a style of software architecture for distributed hypermedia system.
Rotunda	An end-to-end profiler that comes with the Volta installation.
Ruby	Ruby is a dynamic, reflective, general purpose object-oriented programming language.
SaaS	Software as a Service (SaaS) is a software application delivery model where a software vendor develops a web-native software application and hosts and operates (either independently or through a third-party) the application for use by its customers over the Internet. Customers do not pay for owning the software itself but rather for using it.
Script#	A .NET script language aimed at browser programming.
Silverlight	Microsoft Silverlight plug-in for delivering Web User Interfaces.
Solution (Visual Studio)	Set of projects in Visual Studio.
Spoofing	The act of successfully masquerading a person or program as another by falsifying data.
SVG	An XML specification and file format for describing two-dimensional vector graphics.
TFS	Team Foundation Server (commonly abbreviated TFS) is a Microsoft offering for source control, data collection, reporting, and project tracking, and is intended for collaborative software development projects.
Tier	A layer in a multitier software architecture.
Tier-splitting	Splitting of an application among tiers.

URI	Uniform Resource Identifier, an identifier for a resource available on the internet.
URL	Uniform Resource Locator, an URI describing the address of the resource.
Visual Studio	An IDE for .NET application development by Microsoft.
VMware	An application that enables running a virtual machine with its own hardware and software on a host computer.
WCF	Windows Communication Foundation, a Microsoft's framework to build applications that intercommunicate.
WDSL	Web Services Description Language, an XML-based language for describing Web services.
WinForms	A Microsoft's graphical user interface API.
Wireshark	A network protocol analyzer.
XmlHttpRequest	API for transfer of XML and other text data between a web server and a browser.
XNA	A Microsoft's game development platform.

References

1. **PRWeb.** *SaaS 2.0: Saugatuck Study Shows Rapid SaaS Evolution to Business Platforms.* Westport, CT : PRWeb, 2006.

2. **InterAKT.** AJAX: Asynchronously Moving Forward. *InterAKT Online.* [Online] November 10, 2005. [Cited: June 29, 2008.] http://www.interaktonline.com/support/articles/Details/AJAX:+Asynchronously+Moving+Forward-Why+use+AJAX%3F.html?id_art=36&id_asc=309.

3. **Avanade.** Avanade home page. *Avanade.* [Online] Avanade inc., 2008. http://www.avanade.com.

4. *From Sequential Programs to Multi-Tier Applications by Program Transofrmation.* **Matthias Neubauer, Peter Thiemann.** Long Beach, CA : ACM, 2005. 0362-1340.

5. **Microsoft Corporation.** Announcing Volta: Web Development Using Only the Materials in the Room. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 05, 2007. [Cited: February 15, 2008.] http://labs.live.com/volta/blog/Announcing+Volta+Web+Development+Using+Only+The+Materials+In+The+Room.aspx.

6. **Cappell, David.** *Introducing Windows Communication Foundation.* s.l. : Microsoft Corporation, 2007. p. 36.

7. **Avanade.** About Us. *Avanade.* [Online] Avanade inc., 2007. [Cited: March 3, 2008.] http://www.avanade.com/about/.

8. **Microsoft Corporation.** Volta Fundamentals. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 05, 2007. [Cited: February 28, 2008.] http://labs.live.com/volta/docs/.

9. **Dyer, Wes.** Volta Forum - Server to client messaging. *MSDN Forums - Microsoft Live Labs Volta.* [Online] Microsoft Corporation, February 08, 2008. [Cited: April 25, 2008.] http://forums.microsoft.com/msdn/ShowPost.aspx?PostID=2812777&SiteID=1.

10. **Michigan), Thuan L. Thai and Hoang Q. Lam. (University of.** *.NET Framework essentials.* s.l. : O'Reilly, 2003.

11. **Microsoft Corporation.** Frequently Aksed Questions. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation. [Cited: april 10, 2008.] http://labs.live.com/volta/faq/.

12. **Mesbah, Ali.** Scientific Activities. *Ali Mesbah.* [Online] April 16, 2008. [Cited: June 17, 2008.] http://www.st.ewi.tudelft.nl/~mesbah/.

13. **Rotem-Gal-Oz, Arnon.** Fallacies of Distributed Computing Explained. *Arnon Rotem-Gal-Oz's Cirrus Minor.* [Online] June 28, 2006. [Cited: June 25, 2008.] http://www.rgoarchitects.com/Files/fallacies.pdf.

14. **Microsoft Corporation.** UI and Ajax Recipes. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 5, 2007. [Cited: June 9, 2008.] http://labs.live.com/volta/docs/recipes/ux.aspx.

15. **Richter, Christoph.** Volta Forum - Volta & WCF services. *MSDN Forums - Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 10, 2007. [Cited: June 10, 2008.] http://forums.microsoft.com/msdn/ShowPost.aspx?PostID=2532303&SiteID=1.

16. **Fielding, Roy Thomas.** *Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation.* Irvine : University of California, 2000.

17. Comet (programming). *Wikipedia.* [Online] [Cited: 07 09, 2008.] http://en.wikipedia.org/wiki/Comet_(programming).

18. **Microsoft Corporation.** Volta Sample Applications. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 5, 2007. [Cited: 03 29, 2008.] http://labs.live.com/volta/samples.aspx.

19. **Dai Clegg, Richard Barker.** *Case Method Fast-Track: A RAD Approach.* s.l. : Addison-Wesley Professional, 1994. 978-0201624328.

20. **Kantamneni, Harish.** Volta Forum - More integration with ASP.NET. *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, December 19, 2007. [Cited: June 11, 2008.] http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2573924&SiteID=1.

21. **Merrill, Christopher L.** Performance Impacts of AJAX Development. *Web Performance.* [Online] January 15, 2005. [Cited: June 29, 2008.] http://www.webperformanceinc.com/library/reports/AjaxBandwidth/.

22. **Velzen, Danny van.** Volta Forum - Next version of Volta... optimization? *Microsoft Live Labs Volta.* [Online] Microsoft Corporation, March 14, 2008. [Cited: April 20, 2008.] http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=3003469&SiteID=1.

Appendices

A. A very basic chat program

```
using System;
using Microsoft.LiveLabs.Volta.Html;
using Microsoft.LiveLabs.Volta.Xml;

namespace SharedData
{
    public partial class VoltaPage1 : Page
    {
        public Paragraph p1;
        public Paragraph p2;
        public Input invoer;
        public Button get;
        public Button set;
        public Button setname;

        private String name;

        public VoltaPage1()
        {
            InitializeComponent();
            getstring();
        }

        partial void InitializeComponent()
        {
            p1 = new Paragraph();
            p2 = new Paragraph();
            set = new Button();
            get = new Button();
            setname = new Button();
            set.Value = "Send";
            get.Value = "Get new message";
            setname.Value = "Set nickname";
            set.Click += new HtmlEventHandler(setstring);
            get.Click += new HtmlEventHandler(getstring);
            setname.Click += new HtmlEventHandler(setnickname);
            invoer = new Input();

            name = "Unknown";
            Document.Body.Add(p1);
            Document.Body.Add(p2);
            Document.Body.Add(setname);
            Document.Body.Add(invoer);
            Document.Body.Add(set);
            Document.Body.Add(get);
        }
        public void setstring()
        {
            server.setbericht(name + " says: " +
                        invoer.Value.ToString());
            getstring();
        }

        public void getstring()
        {
            p1.InnerText = p2.InnerText;
            p2.InnerText = server.getbericht();
        }

        private void setnickname()
        {
            name = invoer.Value.ToString();
            p1.InnerText = p2.InnerText;
            p2.InnerText = "Nickname set to: " +
                                    invoer.Value.ToString();
        }
    }
}
```

Code sample 7. A very basic chat program.

B. Fibonacci sample program

Program code for calculating the nth Fibonacci number. The calculation can be switched from client to server by uncommenting the `[RunAtOrigin()]` attribute.

```
using System;
using Microsoft.LiveLabs.Volta.Html;
using Microsoft.LiveLabs.Volta.Xml;
using Microsoft.LiveLabs.Volta.MultiTier;

namespace Fibonacci
{
    public partial class VoltaPage1 : Page
    {
        Paragraph p;
        Button b10, b20, b30, b40, b42, b45, b50;

        public VoltaPage1()
        {
            InitializeComponent();
        }

        partial void InitializeComponent()
        {
            p = new Paragraph();

            b10 = new Button();
            b10.Value = "Calculate 10th";
            b10.Click += delegate
            {
                p.InnerText = "result: " + serverside.calculate(10);
            };

            Document.Body.Add(p);
            Document.Body.Add(b10);

            //add buttons for other Fibonacci numbers
        }
    }
```

Code sample 8. A page that outputs the result of Fibonacci calculations.

```
    // currently running at client; uncomment for calculation on server
    // [RunAtOrigin()]
    public static class serverside
    {
        public static int calculate(int order)
        {
            if (order <= 1)
            {
                return order;
            }
            else
            {
                return (calculate(order - 1) + calculate(order - 2));
            }
        }
    }
}
```

Code sample 9. We can decide here whether to compile for client or server tier calculation.

C. Requirements list

1. User Support
    - ~~1.1. Registration~~
    - ~~1.2. Wizard~~
        - ~~1.2.1. Import during wizard~~
            - ~~1.2.1.1. Import from Live account~~
            - ~~1.2.1.2. Import from Facebook~~
            - ~~1.2.1.3. Import from OpenSocial~~
    - 1.3. Login
        - 1.3.1. Login's on other social networks
            - 1.3.1.1. Live Account
            - ~~1.3.1.2. Facebook~~
            - ~~1.3.1.3. OpenSocial~~
2. Profile
    - ~~2.1. Data~~
        - ~~2.1.1. Personal information~~
        - ~~2.1.2. Location~~
    - ~~2.2. Create~~
        - ~~2.2.1. Wizard support~~
    - ~~2.3. Modify~~
        - ~~2.3.1. Automated information updates (i.e. age through birth date)~~
    - ~~2.4. Rights~~
        - ~~2.4.1. View~~
        - ~~2.4.2. Modify by friends~~
        - ~~2.4.3. Group views~~
        - ~~2.4.4. Third party~~
    - ~~2.5. Linked with other social networks~~
        - ~~2.5.1. Live~~
        - ~~2.5.2. Facebook~~
        - ~~2.5.3. OpenSocial~~
    - 2.6. CO2 footprint (linked to 3)
    - 2.7. Buddy list
        - ~~2.7.1. Add, remove, search buddies~~
        - 2.7.2. Statistics
        - ~~2.7.3. Subgroups (i.e. all your school friend)~~
3. CO2 footprint (show/calculation)
    - 3.1. Transportation
        - 3.1.1. Car
            - 3.1.1.1. Petrol
            - 3.1.1.2. Routes
                - ~~3.1.1.2.1. Include extra CO2 according to traffic jams~~
            - ~~3.1.1.3. Carpooling~~
            - ~~3.1.1.4. Last mile~~
        - ~~3.1.2. Public transport~~
            - ~~3.1.2.1. Bus~~
            - ~~3.1.2.2. Train~~

~~3.1.2.3. Metro~~
~~3.1.2.4. Tram~~
~~3.1.3. Boat~~
~~3.1.4. Airplane~~
3.2. Household
3.2.1. Power
3.2.2. Gas
3.2.3. Smart metering
3.3. Food
~~3.4. Products~~
~~3.4.1. Production CO2~~
~~3.4.2. Waste processing~~
~~3.5. Other~~
~~3.5.1. Module support to add new things to measure~~
~~3.5.2. Personal plug-ins~~
3.6. Overview
~~3.6.1. Pie-charts~~
3.6.2. Graphs
3.6.3. View per ~~year, month~~, week ~~and day~~
~~3.6.4. Averages, peaks~~
~~4. Optimization (Goal to lower CO2)~~
~~4.1. Active~~
~~4.1.1. Route planner (TraVolta)~~
~~4.1.1.1. Traffic jam information~~
~~4.1.1.1.1. Predict jams according to route information of our users~~
~~4.1.1.2. Carpooling~~
~~4.1.1.3. Last mile~~
~~4.1.2. Other?~~
~~4.2. Passive (Text tip)~~
~~4.2.1. General random tips~~
~~4.2.2. Specific/Relevant knowhow~~
~~4.2.2.1. Specific knowhow with comparing information~~
5. News
5.1. Show news
~~5.2. News integration from other sources~~
~~5.3. RSS support~~
~~6. Partner management~~
~~6.1. Show list of partners~~
~~6.2. Adding, removing, editing and publishing of GReen partners~~
~~6.3. Banner management~~
~~6.3.1. Show banners~~
~~6.3.1.1. Page dependent banners~~
~~6.3.2. Adding, removing and editing banners~~
7. Mailing list
7.1. User management
~~7.2. Sending emails~~
~~8. Administrative interface~~

~~8.1. Overall right management~~
~~8.2. Web service~~
~~8.2.1. Provide a web service~~
~~8.2.2. Configuration~~
~~8.3. Theme management~~
~~8.4. Content management~~
~~8.5. User management~~

9. Integration with Silverlight

D. MoSCoW analysis

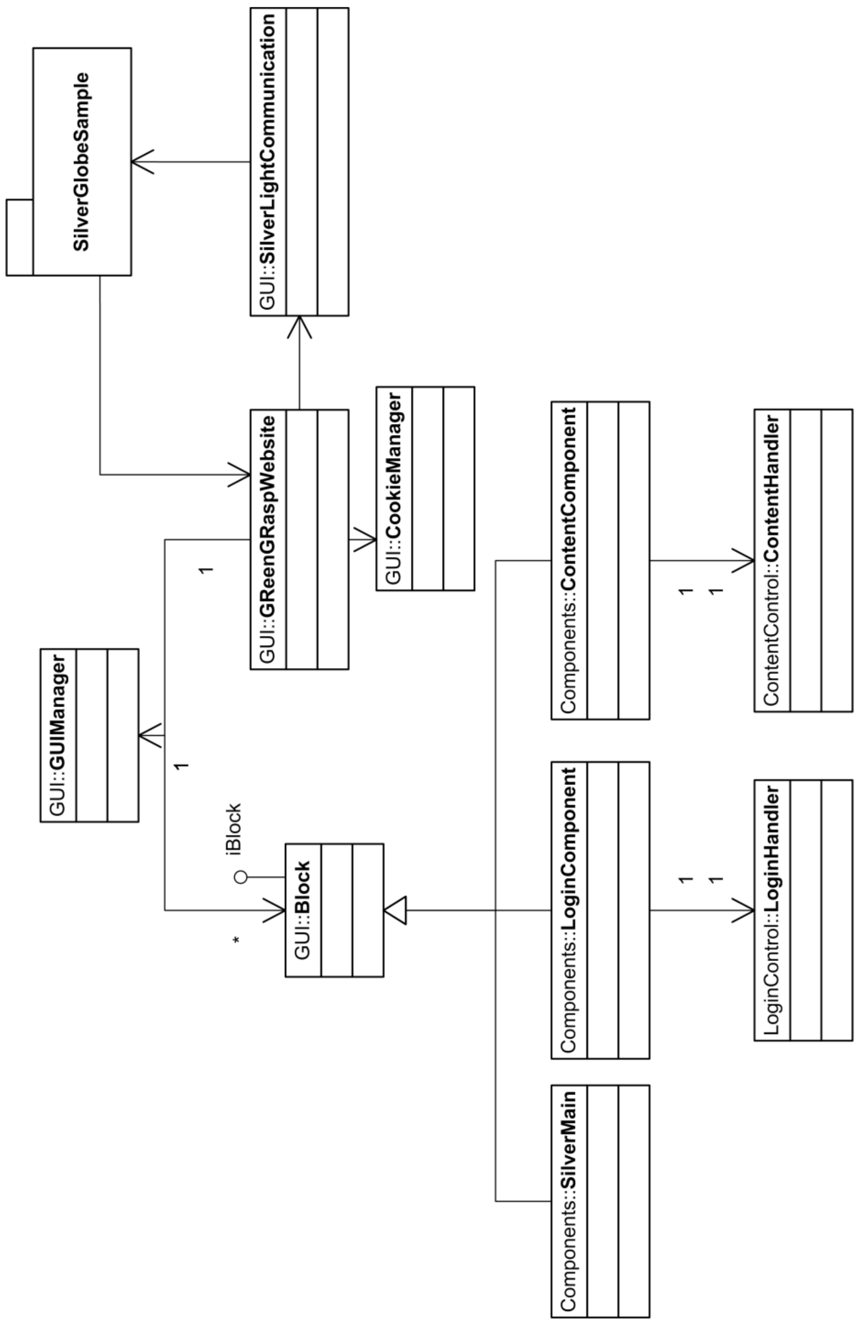
MoSCoW analysis and implementation status

Number	Description	Status
Must haves		
1.	User Support	Implemented
1.1.	Registration	Not implemented: this did not add value to our research or showcase.
1.3.	Login	Implemented
2.7.	Buddy list	Not implemented
2.7.1.	Add, remove, search buddies	Present in Windows Live ID, not incorporated into our site.
3	CO_2-footprint	Implemented as *GReenGRasp* (a positive display of CO_2-footprint)
3.6.	Overview (basic)	Implemented
3.6.3.	View per year, month week and day	An overview of the last five weeks can be displayed
3.6.4.	Averages	Averages per week are calculated as part of the *GReenGRasp.*
4	Optimization (basic)	Not implemented: required too much knowledge regarding CO2 emission, which is not a goal of the project
4.2.1	General tips	
5.	News	Implemented
5.1.	Show news	Implemented
5.2.	News integration from other sources	Not implemented: we already tested the consumption of Web services elsewhere
7.	Mailing list	Implemented, but dropped in the final version, since sending e-mails was not supported by Volta
7.2.	Sending emails	Not Implemented: not possible with Volta

Should haves		
1.2.	Registration Wizard	Not implemented, see feature 1.1
1.2.1.1.	Import from Live account	Idem ditto
1.3.1.1.	Login Live Account	Implemented through ASP.NET
2.	Profile	Implemented in the database and database interface, however nothing is done with it in our current version of the application.
2.1.	Personal information	Not implemented
2.2.	Create Profile	
2.3.	Modify Profile	
2.4.	Rights	
2.4.1.	View	
2.5.	Linked with other social networks	
2.5.1	Live	
2.6.	CO_2 footprint (linked with 3)	Implemented
3.1.	Transportation	Implemented (*TravelGRasp)*
3.1.1.	Car	Implemented
3.1.1.2.	Routes	Implemented through integration with Virtual Earth.
3.1.2.	Public Transport	Not implemented
3.2.	Household	Implemented (*HouseGRasp*)
3.2.1.	Power	Implemented
3.2.2.	Gas	Not implemented
3.3.	Food	Implemented (very basically: *FoodGRasp*)

3.6.	Overview	Implemented
3.6.1.	Pie-charts	Not Implemented: we believed bar and line graphs to provide a better overview
3.6.2.	Graphs	Implemented using *VisiFire* in Silverlight
5.3.	RSS Support	Not implemented, see 5.2
7.1.	User management	
8.	(Basic) administrative interface	
8.1.	General right management	All not implemented: low added value and lack of time
8.4	Content management	
8.5.	User management	
Could haves		
1.2.1.2. Import from Facebook; 1.3.1.2. Login Facebook; 1.2.1.3. OpenSocial; 1.3.1.3. Login OpenSocial; 2.1.2. User Location; 2.5.2. Facebook; 2.5.3. OpenSocial; 3.1.2.1. Bus; 3.1.2.2. Train; 3.1.2.3. Metro; 3.1.2.4. Tram; 3.1.4. Airplane; 3.2.3. Smart Metering; 3.3. Food (more precise); 4.2.2. Specific/Relevant knowhow; 4.2.2.1 Specific knowhow with comparing information; 6. Partner Management; 6.1 Show list of partners; 6.2. Adding, removing, editing and publishing of GReen partners; 6.3. Banner management; 6.3.1. Show banners; 6.3.2. Adding, removing and editing banners; 8. Administrative interface (extended); 8.2. Web service; 8.2.1. Provide a web service; 8.2.2. Configuration		
Would haves		
2.4.2. Modify by friends; 2.4.3. Group views; 2.4.4. Third party; 2.7.3. Subgroups; 3.1.1.1. Petrol; 3.1.1.2.1. Include extra CO_2 according to traffic jams; 3.1.1.3. Carpooling; 3.1.1.4. Last mile; 3.1.3. Boat; 3.4. Product; 3.4.1 Production CO_2; 3.4.2. Waste processing; 3.5. Other; 3.5.1. Module support to add new things to measure; 3.5.2 Personal plug-ins; 6.3.1.1. Page dependent banners; 8.3. Theme management;		

E. Package diagrams



SilverGlobeSample
GUI::SilverLightCommunication
GUI::GReenGRaspWebsite
GUI::CookieManager
GUI::GUIManager
GUI::Block
iBlock
1
1
*
Components::ContentComponent
ContentControl::ContentHandler
1
1
Components::LoginComponent
LoginControl::LoginHandler
1
1
Components::SilverMain

Figure 10. Client package diagram

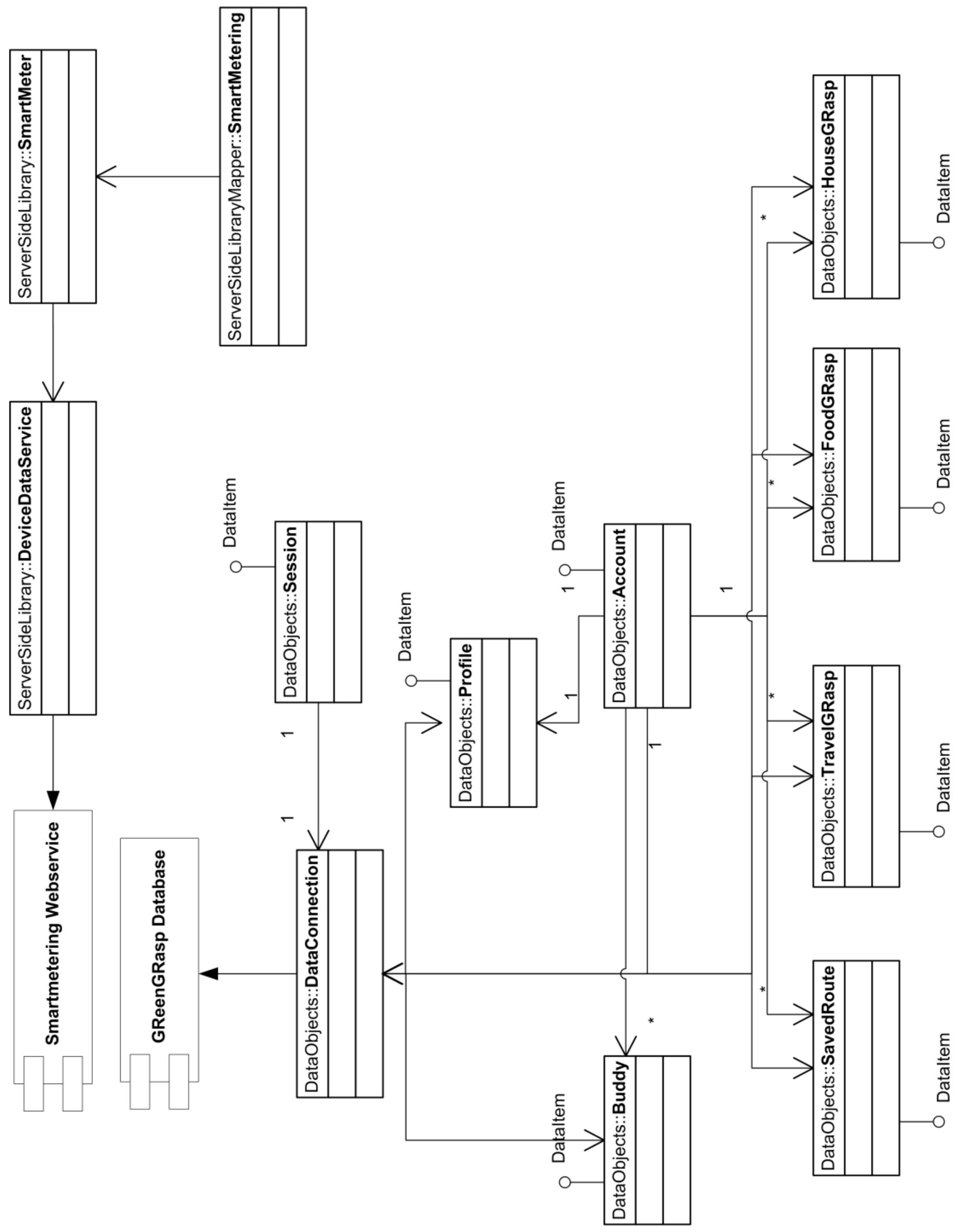


Figure 11. Server package diagram

F. Detailed Class Diagrams per Package

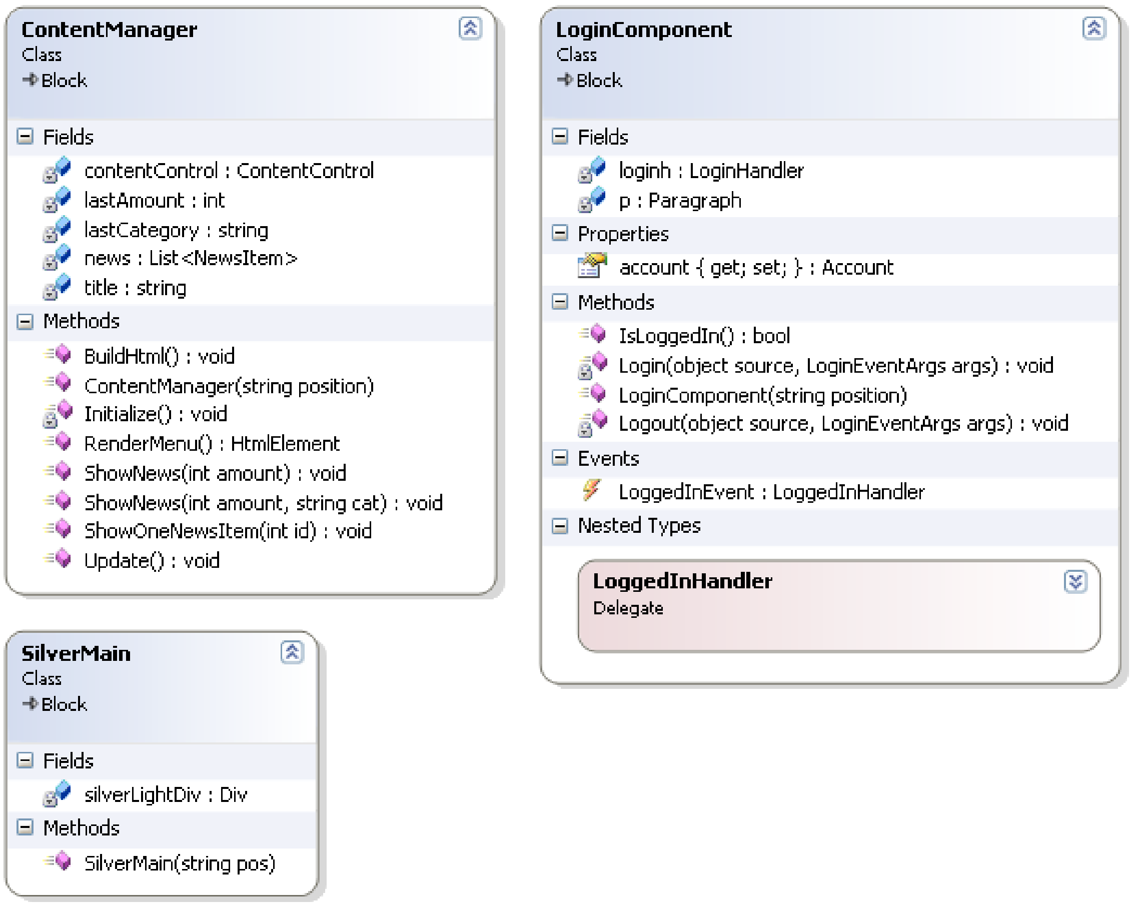
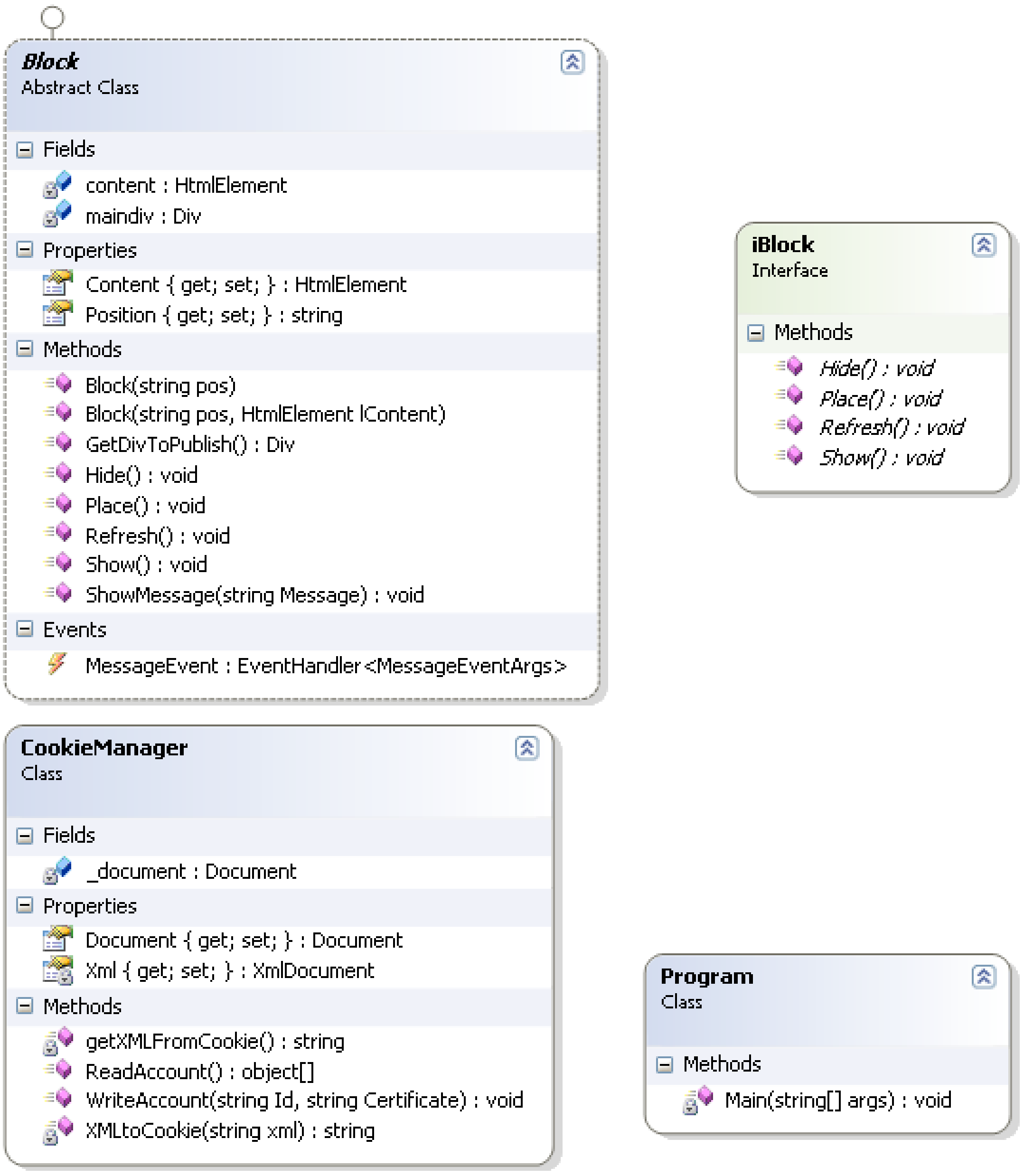
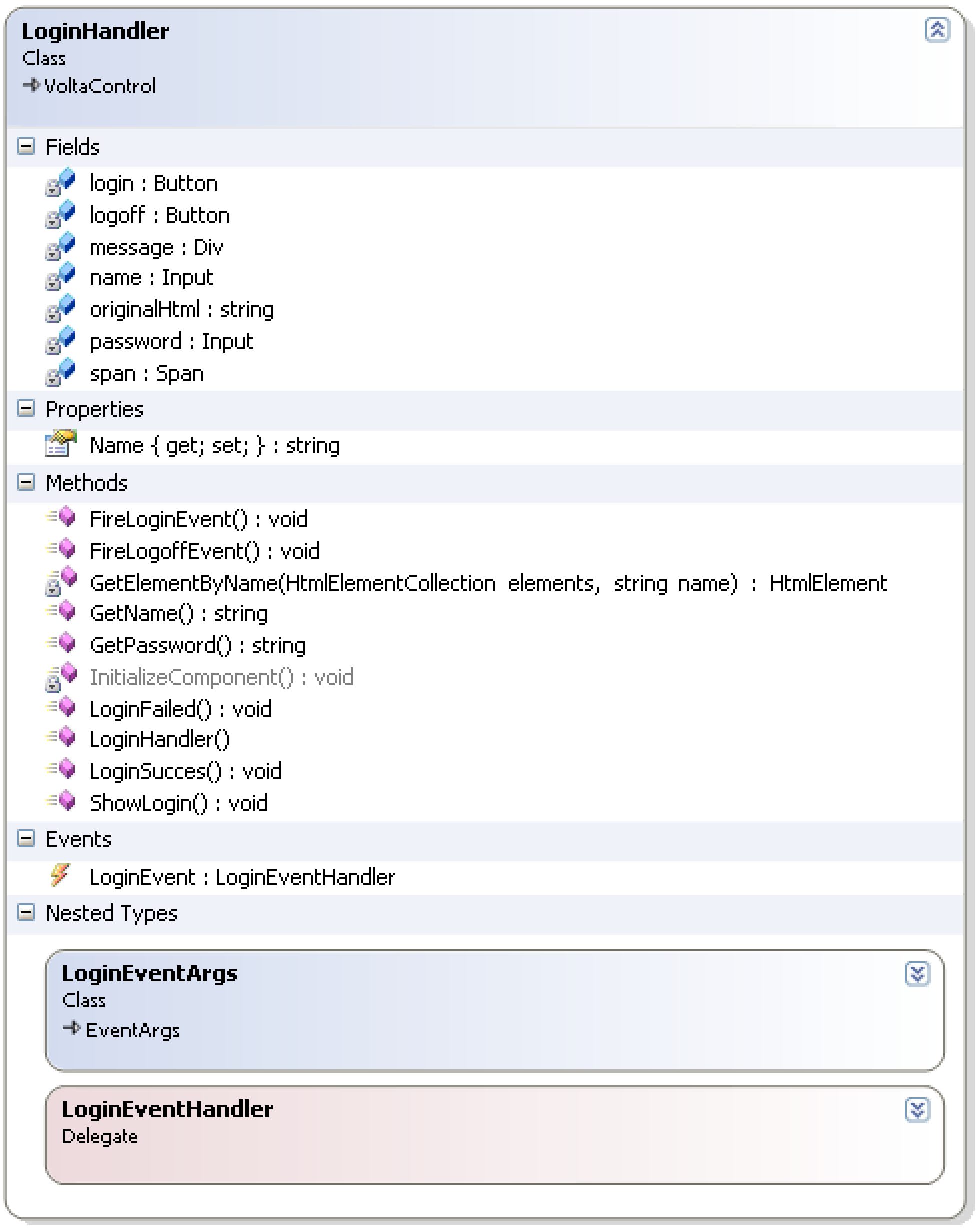


Figure 12. Components package



Block
Abstract Class
Fields
content : HtmlElement
maindiv : Div
Properties
Content { get; set; } : HtmlElement
Position { get; set; } : string
Methods
Block(string pos)
Block(string pos, HtmlElement lContent)
GetDivToPublish() : Div
Hide() : void
Place() : void
Refresh() : void
Show() : void
ShowMessage(string Message) : void
Events
MessageEvent : EventHandler<MessageEventArgs>
iBlock
Interface
Methods
Hide() : void
Place() : void
Refresh() : void
Show() : void
CookieManager
Class
Fields
_document : Document
Properties
Document { get; set; } : Document
Xml { get; set; } : XmlDocument
Methods
getXMLFromCookie() : string
ReadAccount() : object[]
WriteAccount(string Id, string Certificate) : void
XMLtoCookie(string xml) : string
Program
Class
Methods
Main(string[] args) : void

Figure 13. GUI Package



LoginHandler
Class
→VoltaControl

Fields

login : Button
logoff : Button
message : Div
name : Input
originalHtml : string
password : Input
span : Span

Properties

Name { get; set; } : string

Methods

FireLoginEvent() : void
FireLogoffEvent() : void
GetElementByName(HtmlElementCollection elements, string name) : HtmlElement
GetName() : string
GetPassword() : string
InitializeComponent() : void
LoginFailed() : void
LoginHandler()
LoginSucces() : void
ShowLogin() : void

Events

LoginEvent : LoginEventHandler

Nested Types

LoginEventArgs
Class
→EventArgs

LoginEventHandler
Delegate

Figure 14. LoginControl package

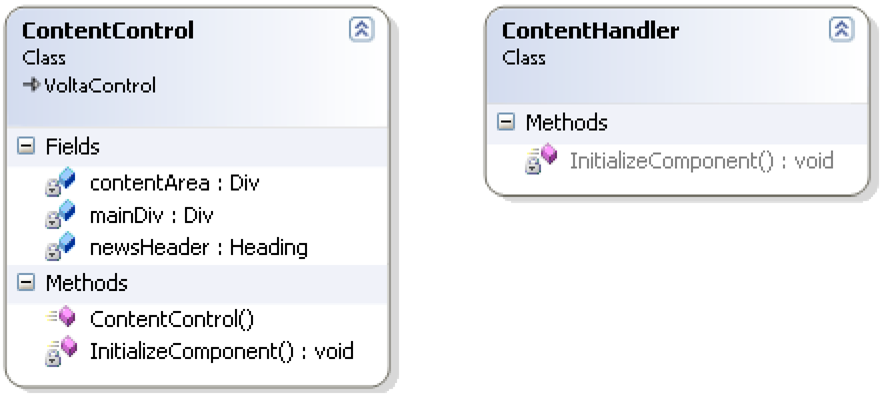


Figure 15. ContentControl package

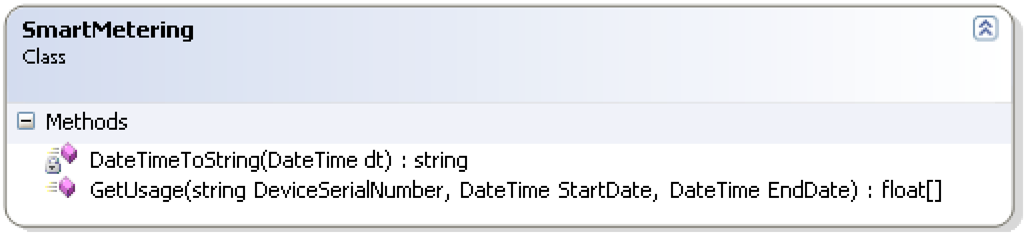

SmartMetering
Class
Methods
DateTimeToString(DateTime dt) : string
GetUsage(string DeviceSerialNumber, DateTime StartDate, DateTime EndDate) : float[]

Figure 16. ServerSideLibraryMapper package

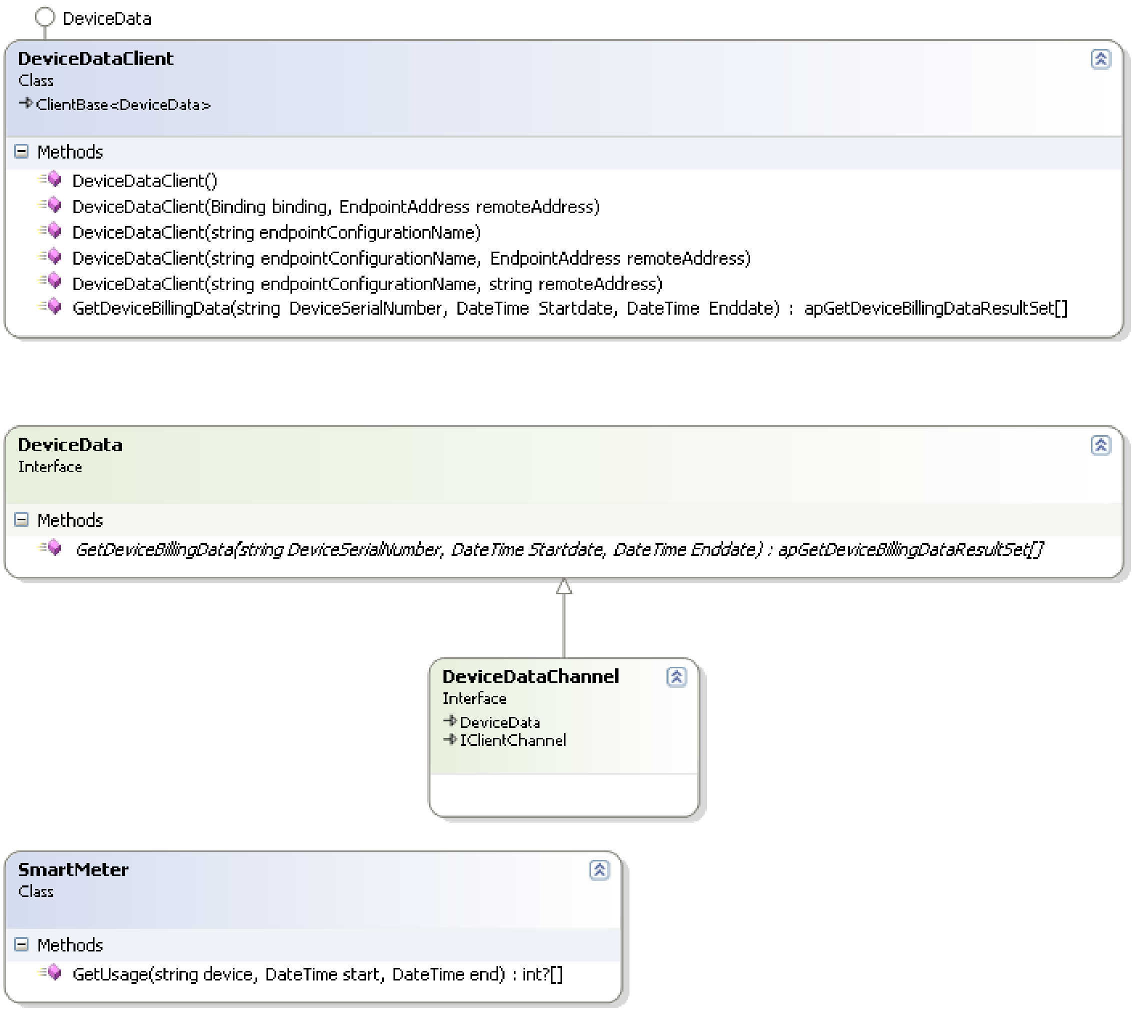

DeviceData
DeviceDataClient
Class
ClientBase<DeviceData>
Methods
DeviceDataClient()
DeviceDataClient(Binding binding, EndpointAddress remoteAddress)
DeviceDataClient(string endpointConfigurationName)
DeviceDataClient(string endpointConfigurationName, EndpointAddress remoteAddress)
DeviceDataClient(string endpointConfigurationName, string remoteAddress)
GetDeviceBillingData(string DeviceSerialNumber, DateTime Startdate, DateTime Enddate) : apGetDeviceBillingDataResultSet[]
DeviceData
Interface
Methods
GetDeviceBillingData(string DeviceSerialNumber, DateTime Startdate, DateTime Enddate) : apGetDeviceBillingDataResultSet[]
DeviceDataChannel
Interface
DeviceData
IClientChannel
SmartMeter
Class
Methods
GetUsage(string device, DateTime start, DateTime end) : int?[]

Figure 17. ServerSideLibrary package

G. Database diagram

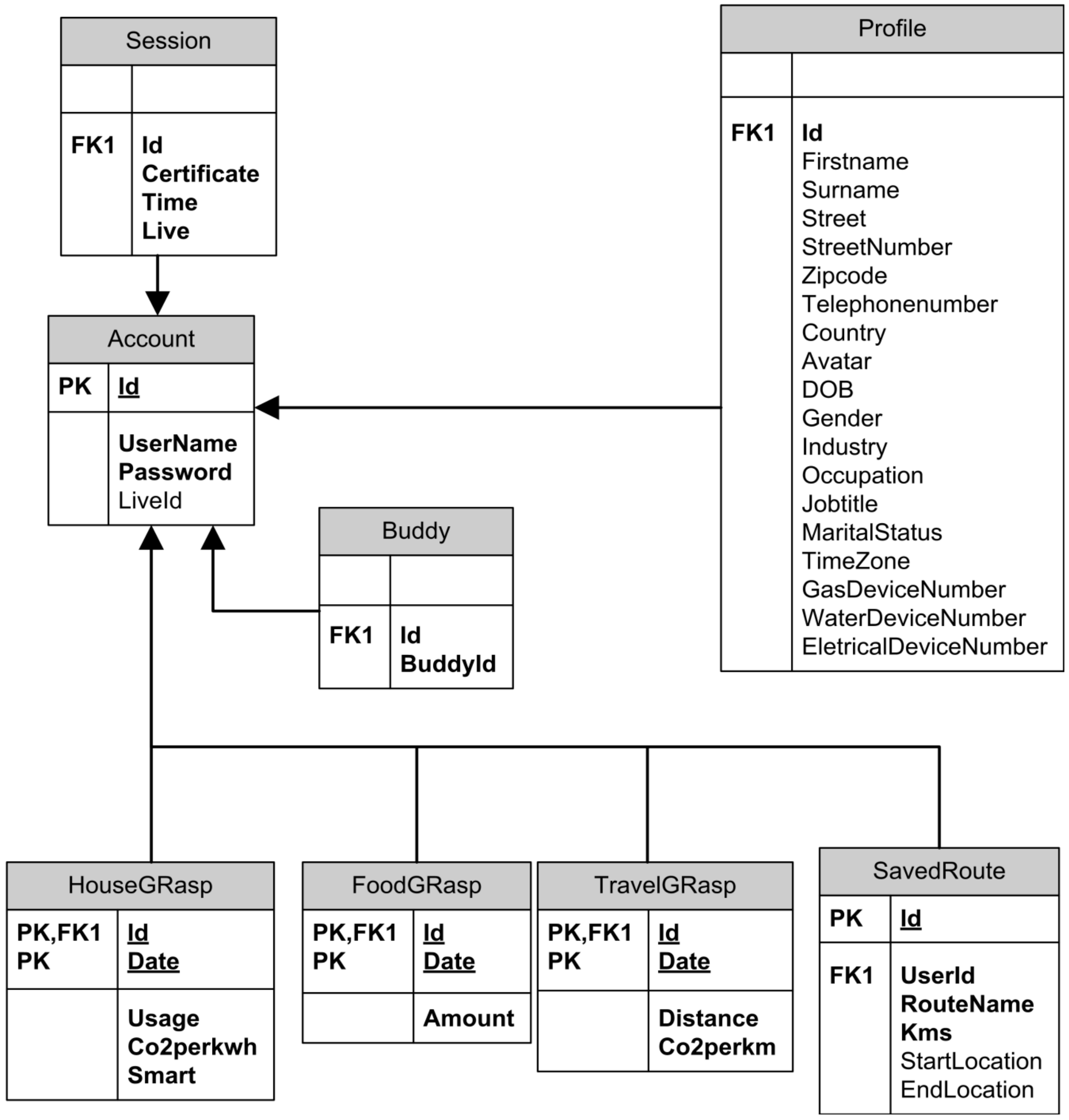


Figure 18. Database Model

H. User Interface GReenGRasp

At the main page the user can see the current news, as present in the News Table of the GReenGRasp Database. On the top the user can login through its GReenGRasp account and on the bottom through Windows Live.

If the login succeeds the user is then presented with the GReenGRasp Silverlight interface. This interface has five main buttons: the Buddy Button, the HouseGRasp button, the Travel GRasp button, the FoodGRasp button and the GReenGRasp display button. Each of these main buttons displays a different side screen, each with different functionalities.

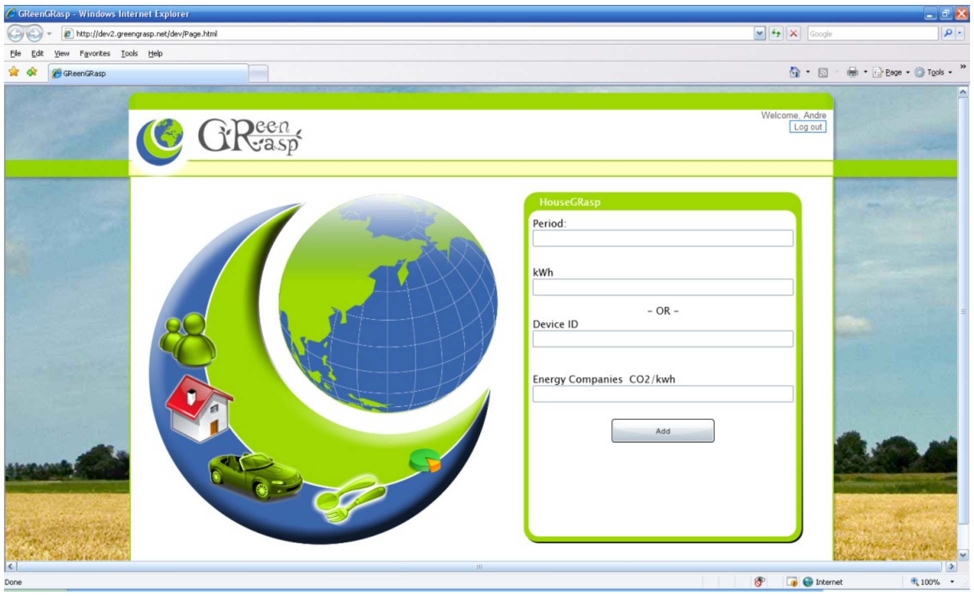
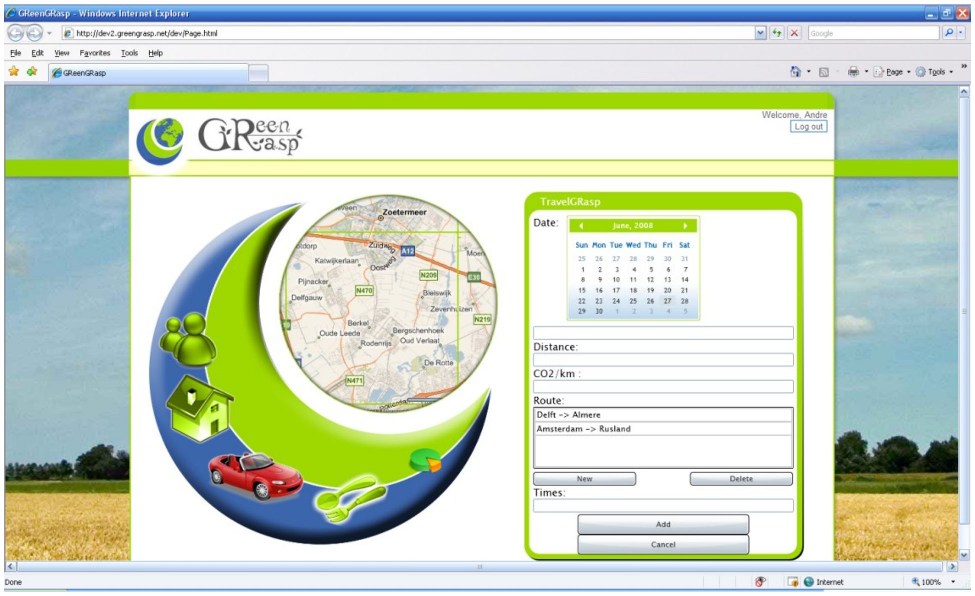


Figure 19. HouseGRasp

The HouseGRasp button displays the interface to update the HouseGRasp. On this interface the user can see the date of their last HouseGRasp update. The date will be displayed in red if the last time the HouseGRasp was updated has surpassed the acceptable interval. This also implies that your HouseGRasp will receive a penalty. Furthermore, the user can update its HouseGRasp using the SmartMetering webservice by supplying the application with his device name. In case the user does not have a smart meter or does not want to use this service, the user can enter the current energy meter standings instead.



Delft -> Almere
Amsterdam -> Rusland

Figure 20. TravelGRasp

The TravelGRasp screen displays an interface to enter a new TravelGRasp entry. A user can select a pre-defined route or define a new one, for which Virtual Earth can be used to calculate the length of this route.

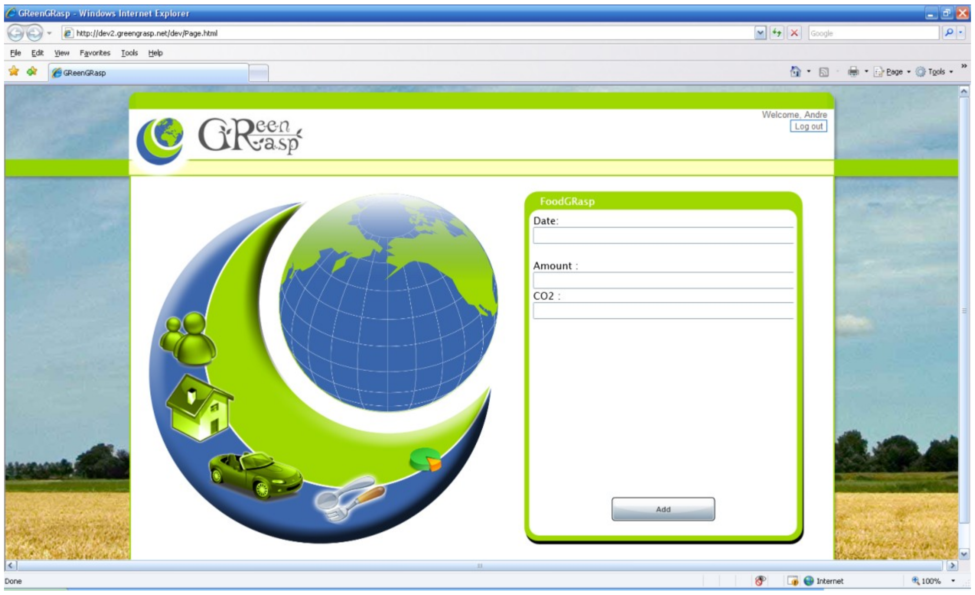

GReenGRasp - Windows Internet Explorer
http://dev2.greengrasp.net/dev/Page.html
GReenGRasp
Welcome, Andre
Log out
FoodGRasp
Date:
Amount :
CO2 :
Add
Done
Internet
100%

Figure 21. FoodGRasp

The FoodGRasp screen displays a simple interface to enter a new FoodGRasp entry; this only consists of a date and an amount of CO2.

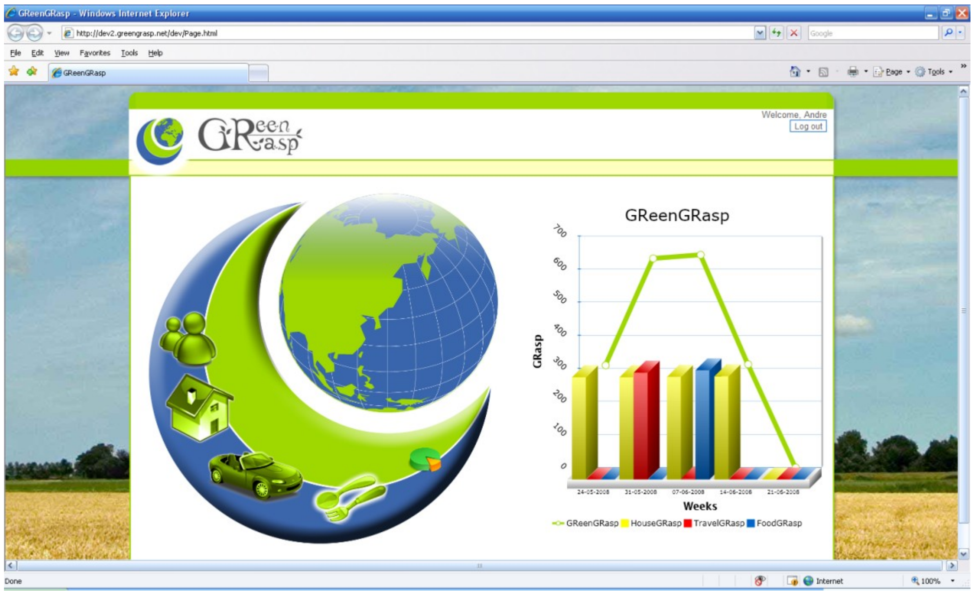


Figure 22. GReenGRasp overview

The GReenGRasp display button creates and shows the GReenChart. This chart displays the three independent GRasps as bars and the composed GReenGRasp as a line for the past 5 weeks.

I. Initial class diagram

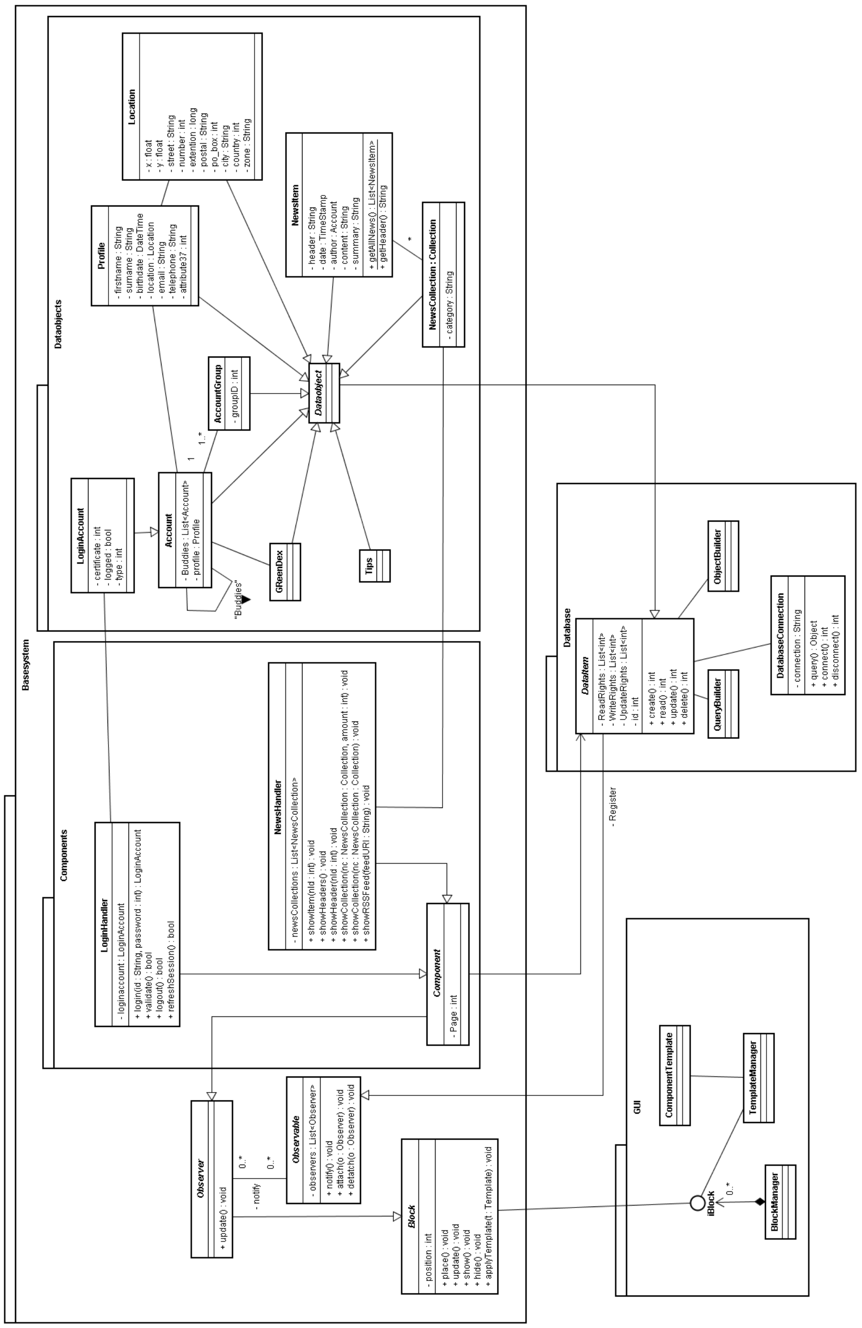


Figure 23. Initial class diagram

J. GReenGRasp

The main aim of GReenGRasp is to deliver a tool where it is possible to supply a concrete indication value on the green awareness of a person. The GReenGRasp value is divided into three main categories: the HouseGRasp, the TravelGRasp and the FoodGRasp.

In and around every house, electricity, gas and water are used to maintain a household. By using new technology, that is already underway, called Smart Metering, we automate the calculation the energy consumption of a household. The GReenGRasp value represents the rate of green awareness of a person. This value should give a recognizable and quick impression to reflect this.

The score is mainly an indication for a user to about how active he or she is in comparison with others. As it becomes more difficult to be more environmentally aware when approaching a neutral state, the GReenGRasp will increase more rapidly when one approaches the neutral area.

A requirement of the GReenGRasp was that it should have a value of 100 when a person's "environmental weight" is on average. The value should be 1000 when a person has no perceivable impact on the environment (this could be accomplished by emission compensation). Another requirement is that it should increase faster when one is able to become relatively "greener" when approaching a neutral environmental weight.

The GReenGRasp function

The following formula satisfies these requirements:

$$G(x) = \frac{1000}{9x + 1}$$

Where *G* represents the GReenGRasp and *x* represents the ratio of the personal vs. average environmental weight (i.e. Carbon Footprint, waste production, wood usage etc).

We use the *ratio* of the environmental weight, because this makes it possible to apply this formula for multiple environmental weight expression methods. This way the formula is generally applicable.

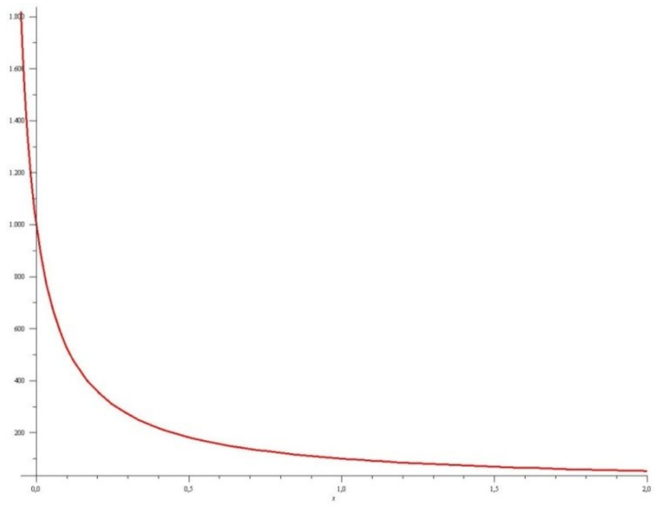
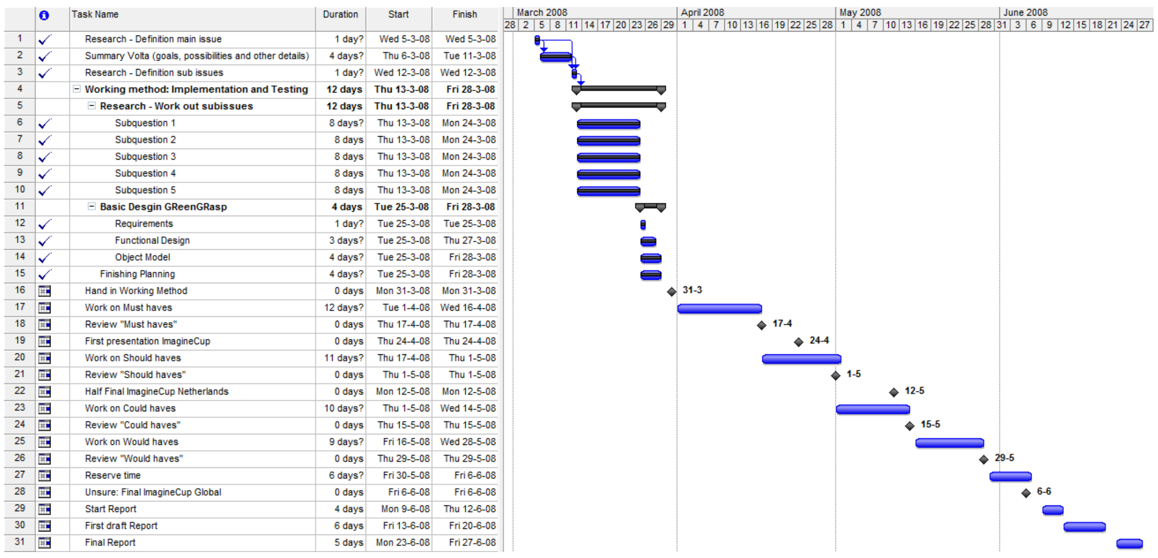


Figure 24. The relation between a person's carbon footprint and GReenGRasp

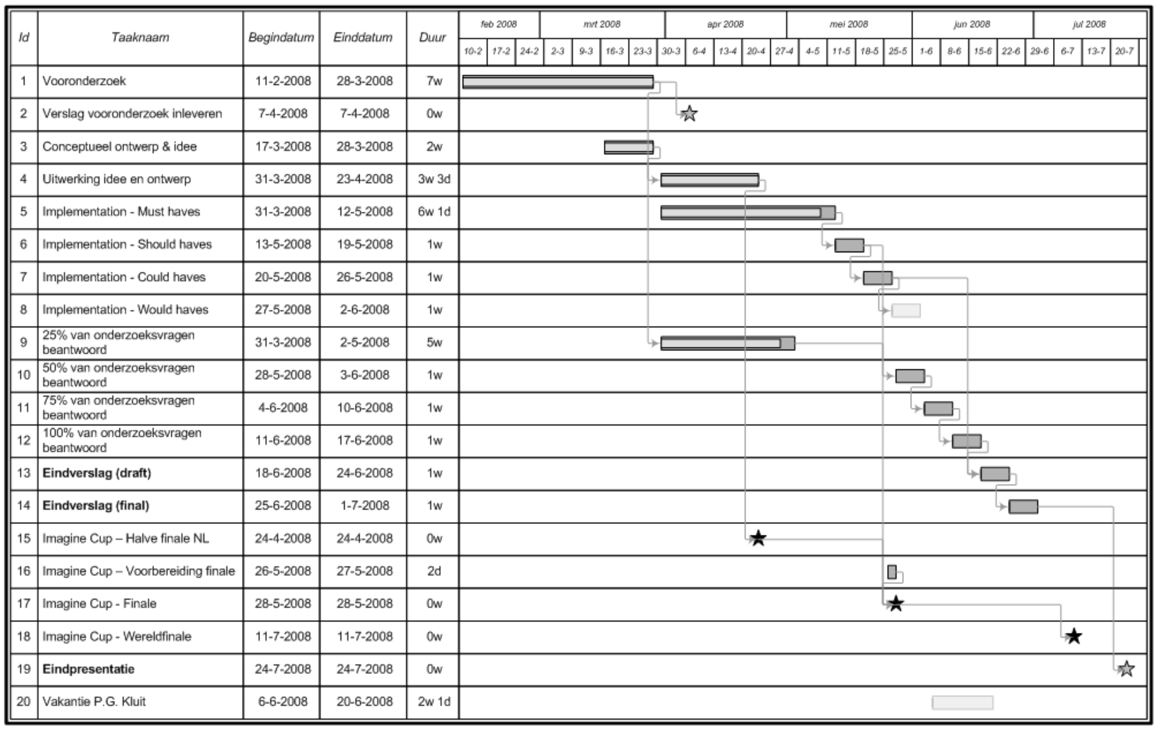
Currently the GReenGRasp is composed by 3 *sub GRaps*: *HouseGRasp*, *TravelGRasp* and *FoodGRasp*. Each of these GRasps contributes equally to the total: the *GReenGRasp*. If no input is provided for a certain GRasp during a certain period a penalty is applied to the score, lowering and sometimes even nullifying a person's GReenGRasp.

K. Planning



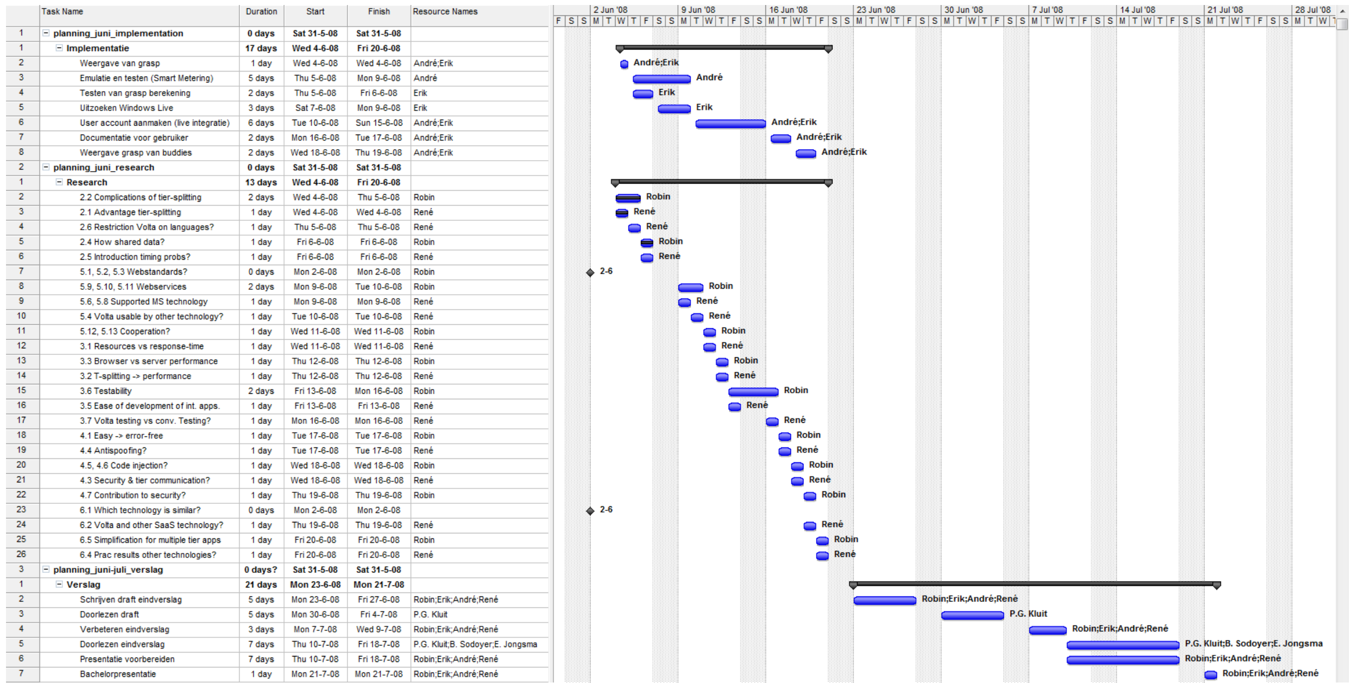
		Task Name	Duration	Start	Finish
1	✓	Research - Definition main issue	1 day?	Wed 5-3-08	Wed 5-3-08
2	✓	Summary Volta (goals, possibilities and other details)	4 days?	Thu 6-3-08	Tue 11-3-08
3	✓	Research - Definition sub issues	1 day?	Wed 12-3-08	Wed 12-3-08
4		**Working method: Implementation and Testing**	**12 days**	**Thu 13-3-08**	**Fri 28-3-08**
5		**Research - Work out subissues**	**12 days**	**Thu 13-3-08**	**Fri 28-3-08**
6	✓	Subquestion 1	8 days?	Thu 13-3-08	Mon 24-3-08
7	✓	Subquestion 2	8 days	Thu 13-3-08	Mon 24-3-08
8	✓	Subquestion 3	8 days	Thu 13-3-08	Mon 24-3-08
9	✓	Subquestion 4	8 days	Thu 13-3-08	Mon 24-3-08
10	✓	Subquestion 5	8 days	Thu 13-3-08	Mon 24-3-08
11		**Basic Desgin GReenGRasp**	**4 days**	**Tue 25-3-08**	**Fri 28-3-08**
12	✓	Requirements	1 day?	Tue 25-3-08	Tue 25-3-08
13	✓	Functional Design	3 days?	Tue 25-3-08	Thu 27-3-08
14	✓	Object Model	4 days?	Tue 25-3-08	Fri 28-3-08
15	✓	Finishing Planning	4 days?	Tue 25-3-08	Fri 28-3-08
16	▦	Hand in Working Method	0 days	Mon 31-3-08	Mon 31-3-08
17	▦	Work on Must haves	12 days?	Tue 1-4-08	Wed 16-4-08
18	▦	Review "Must haves"	0 days	Thu 17-4-08	Thu 17-4-08
19	▦	First presentation ImagineCup	0 days	Thu 24-4-08	Thu 24-4-08
20	▦	Work on Should haves	11 days?	Thu 17-4-08	Thu 1-5-08
21	▦	Review "Should haves"	0 days	Thu 1-5-08	Thu 1-5-08
22	▦	Half Final ImagineCup Netherlands	0 days	Mon 12-5-08	Mon 12-5-08
23	▦	Work on Could haves	10 days?	Thu 1-5-08	Wed 14-5-08
24	▦	Review "Could haves"	0 days	Thu 15-5-08	Thu 15-5-08
25	▦	Work on Would haves	9 days?	Fri 16-5-08	Wed 28-5-08
26	▦	Review "Would haves"	0 days	Thu 29-5-08	Thu 29-5-08
27	▦	Reserve time	6 days?	Fri 30-5-08	Fri 6-6-08
28	▦	Unsure: Final ImagineCup Global	0 days	Fri 6-6-08	Fri 6-6-08
29	▦	Start Report	4 days	Mon 9-6-08	Thu 12-6-08
30	▦	First draft Report	6 days	Fri 13-6-08	Fri 20-6-08
31	▦	Final Report	5 days	Mon 23-6-08	Fri 27-6-08

Figure 25. Planning from the Working Method



Id	Taaknaam	Begindatum	Einddatum	Duur
1	Vooronderzoek	11-2-2008	28-3-2008	7w
2	Verslag vooronderzoek inleveren	7-4-2008	7-4-2008	0w
3	Conceptueel ontwerp & idee	17-3-2008	28-3-2008	2w
4	Uitwerking idee en ontwerp	31-3-2008	23-4-2008	3w 3d
5	Implementation - Must haves	31-3-2008	12-5-2008	6w 1d
6	Implementation - Should haves	13-5-2008	19-5-2008	1w
7	Implementation - Could haves	20-5-2008	26-5-2008	1w
8	Implementation - Would haves	27-5-2008	2-6-2008	1w
9	25% van onderzoeksvragen beantwoord	31-3-2008	2-5-2008	5w
10	50% van onderzoeksvragen beantwoord	28-5-2008	3-6-2008	1w
11	75% van onderzoeksvragen beantwoord	4-6-2008	10-6-2008	1w
12	100% van onderzoeksvragen beantwoord	11-6-2008	17-6-2008	1w
13	**Eindverslag (draft)**	18-6-2008	24-6-2008	1w
14	**Eindverslag (final)**	25-6-2008	1-7-2008	1w
15	Imagine Cup – Halve finale NL	24-4-2008	24-4-2008	0w
16	Imagine Cup – Voorbereiding finale	26-5-2008	27-5-2008	2d
17	Imagine Cup - Finale	28-5-2008	28-5-2008	0w
18	Imagine Cup - Wereldfinale	11-7-2008	11-7-2008	0w
19	**Eindpresentatie**	24-7-2008	24-7-2008	0w
20	Vakantie P.G. Kluit	6-6-2008	20-6-2008	2w 1d

Figure 26. Planning from 8th of May



	Task Name	Duration	Start	Finish	Resource Names
1	**planning_juni_implementation**	**0 days**	**Sat 31-5-08**	**Sat 31-5-08**	
1	**Implementatie**	**17 days**	**Wed 4-6-08**	**Fri 20-6-08**	
2	Weergave van grasp	1 day	Wed 4-6-08	Wed 4-6-08	André;Erik
3	Emulatie en testen (Smart Metering)	5 days	Thu 5-6-08	Mon 9-6-08	André
4	Testen van grasp berekening	2 days	Thu 5-6-08	Fri 6-6-08	Erik
5	Uitzoeken Windows Live	3 days	Sat 7-6-08	Mon 9-6-08	Erik
6	User account aanmaken (live integratie)	6 days	Tue 10-6-08	Sun 15-6-08	André;Erik
7	Documentatie voor gebruiker	2 days	Mon 16-6-08	Tue 17-6-08	André;Erik
8	Weergave grasp van buddies	2 days	Wed 18-6-08	Thu 19-6-08	André;Erik
2	**planning_juni_research**	**0 days**	**Sat 31-5-08**	**Sat 31-5-08**	
1	**Research**	**13 days**	**Wed 4-6-08**	**Fri 20-6-08**	
2	2.2 Complications of tier-splitting	2 days	Wed 4-6-08	Thu 5-6-08	Robin
3	2.1 Advantage tier-splitting	1 day	Wed 4-6-08	Wed 4-6-08	René
4	2.6 Restriction Volta on languages?	1 day	Thu 5-6-08	Thu 5-6-08	René
5	2.4 How shared data?	1 day	Fri 6-6-08	Fri 6-6-08	Robin
6	2.5 Introduction timing probs?	1 day	Fri 6-6-08	Fri 6-6-08	René
7	5.1, 5.2, 5.3 Webstandards?	0 days	Mon 2-6-08	Mon 2-6-08	Robin
8	5.9, 5.10, 5.11 Webservices	2 days	Mon 9-6-08	Tue 10-6-08	Robin
9	5.6, 5.8 Supported MS technology	1 day	Mon 9-6-08	Mon 9-6-08	René
10	5.4 Volta usable by other technology?	1 day	Tue 10-6-08	Tue 10-6-08	René
11	5.12, 5.13 Cooperation?	1 day	Wed 11-6-08	Wed 11-6-08	Robin
12	3.1 Resources vs response-time	1 day	Wed 11-6-08	Wed 11-6-08	René
13	3.3 Browser vs server performance	1 day	Thu 12-6-08	Thu 12-6-08	Robin
14	3.2 T-splitting -> performance	1 day	Thu 12-6-08	Thu 12-6-08	René
15	3.6 Testability	2 days	Fri 13-6-08	Mon 16-6-08	Robin
16	3.5 Ease of development of int. apps.	1 day	Fri 13-6-08	Fri 13-6-08	René
17	3.7 Volta testing vs conv. Testing?	1 day	Mon 16-6-08	Mon 16-6-08	René
18	4.1 Easy -> error-free	1 day	Tue 17-6-08	Tue 17-6-08	Robin
19	4.4 Antispoofing?	1 day	Tue 17-6-08	Tue 17-6-08	René
20	4.5, 4.6 Code injection?	1 day	Wed 18-6-08	Wed 18-6-08	Robin
21	4.3 Security & tier communication?	1 day	Wed 18-6-08	Wed 18-6-08	René
22	4.7 Contribution to security?	1 day	Thu 19-6-08	Thu 19-6-08	Robin
23	6.1 Which technology is similar?	0 days	Mon 2-6-08	Mon 2-6-08	
24	6.2 Volta and other SaaS technology?	1 day	Thu 19-6-08	Thu 19-6-08	René
25	6.5 Simplification for multiple tier apps	1 day	Fri 20-6-08	Fri 20-6-08	Robin
26	6.4 Prac results other technologies?	1 day	Fri 20-6-08	Fri 20-6-08	René
3	**planning_juni-juli_verslag**	**0 days?**	**Sat 31-5-08**	**Sat 31-5-08**	
1	**Verslag**	**21 days**	**Mon 23-6-08**	**Mon 21-7-08**	
2	Schrijven draft eindverslag	5 days	Mon 23-6-08	Fri 27-6-08	Robin;Erik;André;René
3	Doorlezen draft	5 days	Mon 30-6-08	Fri 4-7-08	P.G. Kluit
4	Verbeteren eindverslag	3 days	Mon 7-7-08	Wed 9-7-08	Robin;Erik;André;René
5	Doorlezen eindverslag	7 days	Thu 10-7-08	Fri 18-7-08	P.G. Kluit;B. Sodoyer;E. Jongsma
6	Presentatie voorbereiden	7 days	Thu 10-7-08	Fri 18-7-08	Robin;Erik;André;René
7	Bachelorpresentatie	1 day	Mon 21-7-08	Mon 21-7-08	Robin;Erik;André;René

Figure 27. Planning from the 4th of June

L. Server specifications

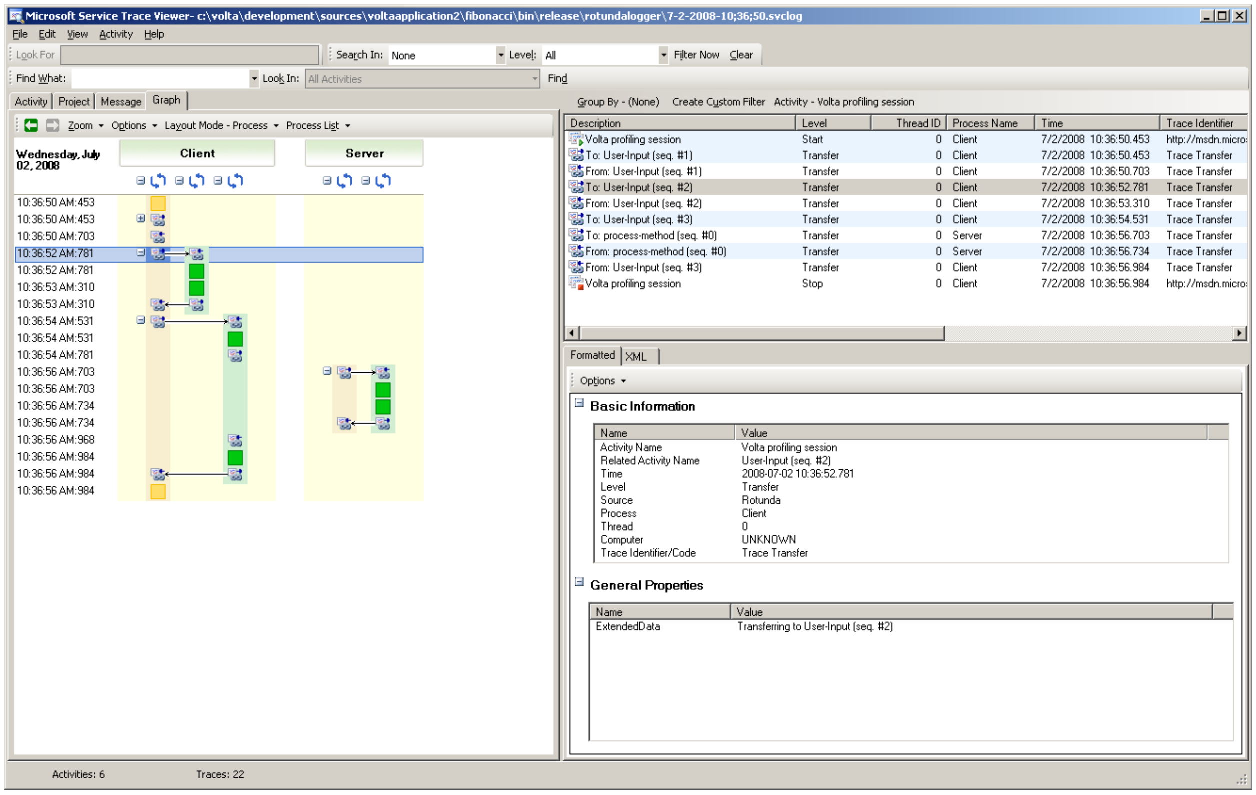
Virtual Hardware (VMware 1.05)	
Processor	Intel® Pentium® M processor, 2.00 GHz, 32-bit
Memory	1022 MB
Hard disk drive size	16 GB
Network interface	1 Gbit/s
Software	
Operating System	Windows Web Server 2008
Web server	Internet Information System 7.0
Database	MS SQL Server 2005
IDE	Visual Studio 2008 Team Foundation edition
Design Environment	Microsoft Blend 2.5 (March and June preview)
Other	.NET framework 3.5
	Volta CTP release of February 21st
	SQL Management Studio 2005
	Silverlight 2 beta 1 and beta 2

M. Rotunda end-to-end profiling

Rotunda is a project by Microsoft that can profile end-to-end performance of applications. The goal is to enable developers to create more efficient applications for the Web tier.

A first call which runs client-side can be seen on the left side of the picture, between 10:36:52:AM:781 and 10:36:53:AM:310.

The operations between 10:36:54:AM:531 and 10:36:56:AM:984 can clearly be distinct as a call to a server-side function.


Microsoft Service Trace Viewer- c:\volta\development\sources\voltaapplication2\fibonacci\bin\release\rotundalogger\7-2-2008-10;36;50.svclog
File Edit View Activity Help
Look For
Search In: None
Level: All
Filter Now Clear
Find What:
Look In: All Activities
Find
Activity Project Message Graph
Zoom Options Layout Mode - Process Process List
Group By - (None) Create Custom Filter Activity - Volta profiling session
Wednesday, July 02, 2008
Client
Server
Formatted XML
Options
Activities: 6
Traces: 22

```
10:36:50 AM:453
10:36:50 AM:453
10:36:50 AM:703
10:36:52 AM:781
10:36:52 AM:781
10:36:53 AM:310
10:36:53 AM:310
10:36:54 AM:531
10:36:54 AM:531
10:36:54 AM:781
10:36:56 AM:703
10:36:56 AM:703
10:36:56 AM:734
10:36:56 AM:734
10:36:56 AM:968
10:36:56 AM:984
10:36:56 AM:984
10:36:56 AM:984
```

Description	Level	Thread ID	Process Name	Time	Trace Identifier
Volta profiling session	Start	0	Client	7/2/2008 10:36:50.453	http://msdn.micro:
To: User-Input (seq. #1)	Transfer	0	Client	7/2/2008 10:36:50.453	Trace Transfer
From: User-Input (seq. #1)	Transfer	0	Client	7/2/2008 10:36:50.703	Trace Transfer
To: User-Input (seq. #2)	Transfer	0	Client	7/2/2008 10:36:52.781	Trace Transfer
From: User-Input (seq. #2)	Transfer	0	Client	7/2/2008 10:36:53.310	Trace Transfer
To: User-Input (seq. #3)	Transfer	0	Client	7/2/2008 10:36:54.531	Trace Transfer
To: process-method (seq. #0)	Transfer	0	Server	7/2/2008 10:36:56.703	Trace Transfer
From: process-method (seq. #0)	Transfer	0	Server	7/2/2008 10:36:56.734	Trace Transfer
From: User-Input (seq. #3)	Transfer	0	Client	7/2/2008 10:36:56.984	Trace Transfer
Volta profiling session	Stop	0	Client	7/2/2008 10:36:56.984	http://msdn.micro:

Basic Information

Name	Value
Activity Name	Volta profiling session
Related Activity Name	User-Input (seq. #2)
Time	2008-07-02 10:36:52.781
Level	Transfer
Source	Rotunda
Process	Client
Thread	0
Computer	UNKNOWN
Trace Identifier/Code	Trace Transfer

General Properties

Name	Value
ExtendedData	Transferring to User-Input (seq. #2)

Figure 28. A screenshot from a client-side and server-side Fibonacci calculation

N. Silverlight implementation

From Silverlight to Volta we use the `HtmlPage.Window.Invoke` method, as can be seen in Code sample 10. This invokes a JavaScript-function that is exported by the `GReenGRaspWebsite` class in Volta.

```
public void RouteSelected(object sender, EventArgs e)
{
    string name =
      (string)((ListBox)Routes.FindName("SavedRoutes")).SelectedItem;

    HtmlPage.Window.Invoke("getroutedistance", new object[]
        {
            name
        });
}
```

Code sample 10. Silverlight to Volta communication

Furthermore, since the exported functions must be static, using static events to link these functions to the non-static methods in our `GReenGRaspWebsite` was necessary. Also, all exported function names have to be lowercase; this was discovered through trial and error. Code sample 11 shows an example of this construction applied to a route function.

```
[Export]
public static void getroutedistance(string name)
{
    // Fire getRouteDistanceEvent with name in the argument
    getRouteDistanceEvent(null,
        new SilverLightCallEventArgs(new object[]
            {
                name
            }));
}
```

Code sample 11. Silverlight to Volta communication in GReenGRaspWebsite

Communication from Volta to Silverlight works differently. On the Volta side we use the browser class `SilverLightCommunication` to import JavaScript functions defined in *Page.html*, as can be seen in Code sample 12.

```
[Import("createChart", PassTypeParametersAsArguments = false)]
public extern void createChart(Array<string> xvalues,
                                Array<string> yvalues);
```

Code sample 12. Importing a function to `SilverLightCommunication`

These JavaScript functions call methods of the Silverlight application marked with the *ScriptableMember*-attribute. A sample of such a JavaScript function can be seen in Code sample 13; it calls the function of Code sample 14.

```
function createChart(a, xvalues, yvalues)
{
    var silverLightControl =
        document.getElementById('greensilver');
    silverLightControl.Content.Page.createChart(xvalues,yvalues);
}
```

Code sample 13. Calling a method of the Silverlight application in the JavaScript of Page.html

In order to send arrays through this path, the Volta `JavaScript.Array` class must be used, which is interpreted as a `ScriptObject` by Silverlight and can be converted to a C# array. This is also displayed in Code sample 14.

```
[ScriptableMember]
public void createChart(object xValues,object yValues)
{
    string[] x = ((ScriptObject)xValues).ConvertTo<string[]>();
    string[] y = ((ScriptObject)yValues).ConvertTo<string[]>();
    …
}
```

Code sample 14. Page.xaml.cs in SilverLight

O. Pre-research report