DEPARTMENT OF SOFTWARE ENGINEERING (EEMCS)
DELFT UNIVERSITY OF TECHNOLOGY

BuddyFuse

Bachelor project (IN3405)

Yousef El-Dardiry (1332686)

1/26/2009

Summary

BuddyFuse is a software product that seamlessly integrates the Dutch social network Hyves into the popular instant messaging client Windows Live Messenger. It aims to enable its users to interact with all of his friends from an application he is already familiar with. The product has been developed as a bachelor graduation project at the Delft University of Technology.

The project started off by conducting a research study. During this study, three questions have been answered to establish a solid base for the rest of the project. In this study an agile software development methodology has been chosen for use in the project. The main technological challenge of the project has been to extend a proprietary software application with additional functionality. The concept of function interception has been investigated extensively in the research study and serves as a basis to overcome this challenge. Furthermore, so-called Application Programming Interfaces (APIs) have been explored to see what methods of integration Windows Live Messenger and Hyves natively expose to third party developers.

After the requirements of the project had been agreed upon, time was spent on the high-level design of the application. This has led to a well thought through and documented architectural design, on which the development iterations have been designed and implemented.

This document serves as an introductory overview of the entire project. After a brief introduction into the ideas behind the application, the project plan and description are being discussed. Afterwards, all different phases and continuous processes such as quality assurance of the project are described. Several references are made to the original documents describing each phase, which have been included as appendices. At the end, I conclude with a personal reflection on the project and my perspectives for this product in the future.

The result of this project is that at the moment of writing all essential features have been implemented and tested. For a quick impression one is encouraged to take a look at the image walkthrough (APPENDIX F). The application is currently in a private beta test and scheduled to be released in the first quarter of this year.

Preface

This document gives an overview of my bachelor graduation project at the Delft University of Technology. The project is about an idea for a software product I have had in mind since the beginning of 2008. I always have had a number of ideas for new products, which is the main reason I have always been attracted to computer science; it enables one to quickly turn ideas into actual products relative to other fields of science.

The idea that I worked on during this project, is to integrate the Dutch social network Hyves into Windows Live Messenger. Social media has always been one of my primary fields of interest, and this idea obviously fits well into that category. I also have a lot of experience developing on the Windows Live platform, for which I have been awarded the Microsoft Most Valuable Professional award in 2007 and 2008. I also worked at the Windows Live Platform Incubation team for three months as a summer intern at the Microsoft Corporation headquarters in Redmond, USA. I would like to thank the people at Microsoft for playing a major role in my education as a software engineer and the opportunities, inspiration and experiences they have given to me.

During the course of this project I have had assistance from a number of people I would like to thank. I developed the project together with Jeroen Bransen with whom I discussed about every aspect of it; the project would not be where it is now without his assistance. Michel van den Berg has been of great assistance during the initial software design phase of the project. I would like to thank the study advisors and my mentor Bernard Sodoyer from the Delft University of Technology for the great flexibility they have shown allowing me to use this project in my curriculum. I would also like to thank the latter for guiding me throughout the project and regularly reviewing every aspect of it. Last but not least, my parents and brothers have been of great assistance not only by reviewing the documentation, but also by guiding me through the months spent on the project personally.

Although I have successfully taken part in a number of different software development competitions with my concepts over the last five years, most of the ideas have never been completely finished. By working on this project as a university project I hope to have structured it in a way it can fully live up to its potential.

Yousef El-Dardiry
Delft, the Netherlands
December 15th, 2008

Table of Contents

1 Introduction

Nowadays, an increasing number of computer applications are trying to enhance the way people interact with each other. Social networks have significantly changed the way their users interact with their friends. At the core of all these social applications (social media related software) lies the social graph; a graph representing individuals and their relations with each other. It can be seen as the data social applications are built upon. Brad Fitzpatrick was one of the first to use the term social graph in (Fitzpatrick & Recordon, 2007), where he explains his thoughts on various problems social applications are facing.

Perhaps in an ideal world, there would be a single comprehensive representation of the social graph easily accessible by any application. The social applications built on top of it would be interacting with the same graph and therefore indirectly interacting with each other. However, the reality is nowadays there are hundreds of different (incomplete) social graphs, built on different technologies and often difficultly accessible by third party applications.

A number of initiatives have been started to tackle this problem. Over the last few years, we have seen many large social networks exposing data about their users via Application Programming Interfaces (APIs) to third parties. In other words, the walls around the different social graph silos are slowly being torn down.

A lot of mashups (a web application hybrid combining data from different sources) have been created adding functionality on top of the exposed data. However, the APIs available are still very diverse across different services, making it a cumbersome task to truly interconnect different social networks.

Besides the technical difficulties, often there is not enough business value for companies in integrating their own social graph with another company's. Even when standard data formats would have been agreed upon and costs to implement interconnections between the different social applications would be irrelevant, it is doubtful a single comprehensive social graph will ever be created. After all, a company creates value by differentiating its products from its competitors'.

Although this makes sense from their perspective, it results in an end user having to enter the same information in different social applications again and again. Next to the duplicate task of entering information, he also is required to access different applications to interact with different parts of his social graph, in a way specific to that particular application.

The goal of this project is to integrate different social applications. However, as opposed to creating another social application aggregating data from different services, the goal is to pro-actively integrate one social application into another. Using the APIs exposed by different services where possible, we want to bring the functionality exposed by different social networks to different parts of the user's computer experience.

1.1 Project description

Because the above problem statement and goal definition remains somewhat vague, it is important to narrow the scope of the project. The first version will consist of an integration of the Dutch social network Hyves into Windows Live Messenger. Because of the large overlapping user base (most users on Hyves also use Windows Live Messenger)

this seems to be a good starting point. Before continuing discussing different aspects of this project, let us briefly introduce both services.

1.1.1 Hyves

Since its release in 2004 the social network Hyves of Dutch origin has quickly grown to the most popular social network in its home country (more than six million users) (NU.nl/Wieland van Dijk, 2007). Creating connections and staying in contact with friends is the main focus of the website. The latest developments include support for instant messaging and the availability of APIs third parties can leverage.

1.1.2 Windows Live Messenger

Windows Live Messenger is an instant messaging application developed by Microsoft. Formerly known as MSN Messenger, it is currently the most popular instant messaging client in the Netherlands, and one of the most popular worldwide. Over the years, many features have been added to the application, including a different number of APIs.

Social networks and instant messaging applications are closely related to each other. After all, they are all built upon information from a user's social graph. This project will focus on integrating features from Hyves into Windows Live Messenger in the form of an extension program built on top of Messenger. The main requirement of this integration is that it must be realized in a seamless way. In other words; the end-user should barely notice he is using an additional add-on on top of the Messenger client he is already familiar with. The original Dutch project description has been included as APPENDIX A.

1.2 Project plan

The PROJECT PLAN (APPENDIX B) introduces the basic ideas by which the project will be executed. The project consists of a number of different phases. In the research phase, a number of research questions will be answered to ensure there is enough knowledge on these areas to continue with the other phases. Before it is possible to start with the design and implementation of the project, it is important to have all requirements documented. This is the goal of the requirements phase.

Based on the software development method chosen in the research phase, the design and implementation phase will embark. In this phase the application will be designed, built and tested.

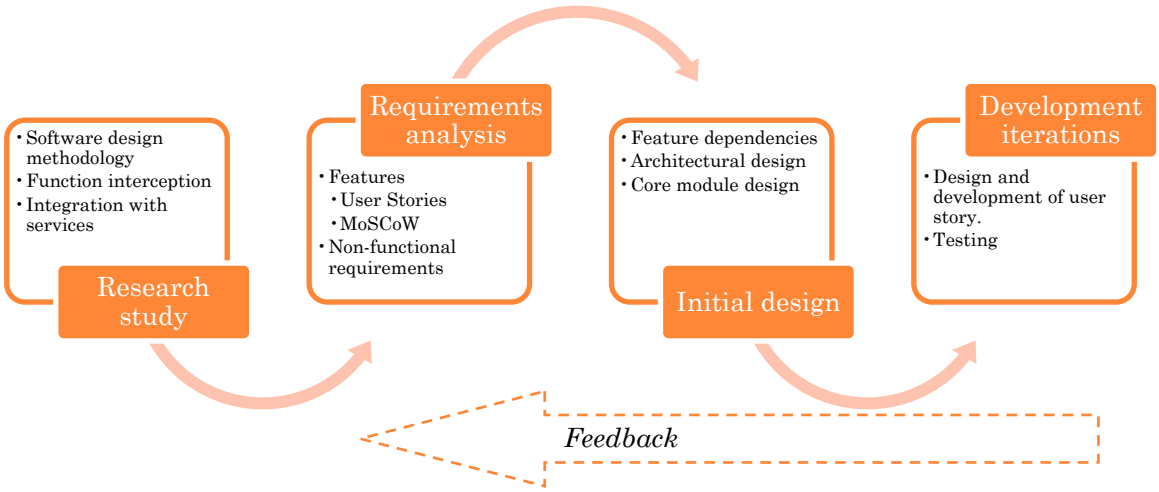

Figure 1-1: overview of the project phases

Figure 1-1 shows an overview of the different project phases. The last phase (design and implementation) has been split up between initial design and the actual development iterations. The feedback arrow indicates how new experiences during later phases have been incorporated into previous documents.

The project plan also specifies the tools to be used and guidelines to be followed, the final project deliverables and introduces the people involved in the project (the project team consists of two developers). Also, the project plan contains a planning continuously updated throughout the project.

2 Project execution

This chapter will covers the execution of the project. It discusses the different phases of the project and summarizes the results of every phase.

2.1 Research

About the first three weeks of the project were spent on conducting a research study. In this study, three questions have been answered to establish a solid base for the rest of the project:

- Which software development methodology is most applicable for use in this project?
- How can one add functionalities to existing compiled programs beyond the possibilities of exposed extensibility-APIs and without access to its source code?
- What are the technologies behind Messenger and Hyves, and what kind of APIs do these services expose?

Below the answers to these questions as found in the RESEARCH STUDY (APPENDIX C) are summarized.

2.1.1 Software development methodology

The goal of this research study was to choose a suitable software development methodology for the project. The traditional waterfall model and two models in the agile programming family have been taken into consideration, namely extreme programming and feature driven development.

The pros and cons of every methodology have been taken into account and compared to each other. The conclusions of this comparison have been projected onto the nature of the project, and based on this a suitable methodology was chosen.

The waterfall model did not seem to be suitable for use because of the possibly changing requirements. Next to that, it would have been hard to plan for technically difficult areas in the design that might require prototyping up front.

Eventually, the extreme programming methodology was chosen with a couple of adjustments. The user stories should be divided into different domains, similar as how features are categorized in feature driven development. Before starting on a task

belonging to a particular domain, that domain should be properly designed and an underlying framework required by all the tasks belonging to the domain should be built first. The global design of the different domains should be well documented. Just like in feature driven development, a task should be completely tested before moving on to the next task.

2.1.2 Integrating with existing applications

Without the source code of an existing application / program, it is a cumbersome task to add new functionality to it (beyond the possibilities exposed by APIs). For this project Windows Live Messenger has to be extended with new buttons, contacts and other functionality. The goal of the second part of the research study was to explain the concept of function interception by which this can be accomplished.

In the study, three different methods to implement function interception are being explained and compared (proxy libraries, import address table modification and in-memory function modification). Since two of these methods require the intercepting function to modify the memory of the target application, first the concept of DLL injection is being introduced. By using DLL injection, the intercepting function can be executed in the target process and therefore modify the memory of this target process.

After explaining the ideas and techniques behind function interception, a simple application of the method is given as an example. We explain how function interception can be used to monitor and modify the network traffic of an application by intercepting Winsock and WinInet functions.

2.1.3 Interaction with applications

Since the application being built has to interact with two other programs / services, it is important to give an introduction to these programs and the way third party programs can interact with them. In the last part of the research study, we take a closer look at both Windows Live Messenger and Hyves, the technologies they are built upon, and the Application Programming Interfaces (APIs) they expose for third party applications.

2.1.3.1 Windows Live Messenger

Although Windows Live Messenger is proprietary software and its source code is not publicly available, there are a number of possibilities to integrate either with the software directly via APIs, or via the technologies it is built upon. The APIs exposed by Windows Live Messenger do not appear to expose the advanced level of functionality required for this project. Mainly, the APIs allow a third party to extend existing scenarios of the application (such as adding a game or a chat robot). Next to the Windows Live Messenger APIs themselves, the Windows Live Platform exposes a number of different APIs to talk to the data source (social graph) Messenger has been built upon.

The protocol used by Windows Live Messenger to communicate with the Microsoft chat servers is TCP based and known as MSNP. Specifics about the protocol have been investigated by hobbyists and made available on the internet (MSNPiki), (Mintz). Windows Live Messenger also uses a proprietary UI framework called Direct UI, about which not much information is available.

2.1.3.2 Hyves

Hyves is a web application and does not run on the desktop of the end user. This means the technology used in their platform will mostly be irrelevant for us, since we will not be able to integrate with these technologies directly. We can only interact with the APIs exposed by the service. Hyves currently exposes three different ways to third parties to interact with its service, namely their web based API, chat platform and gadget platform.

The web based API exposes methods to interact with Hyves on behalf of the end user. The chat platform used by Hyves is built upon open standards and also accessible by third parties. The Hyves gadget platform enables developers to create small gadgets Hyves users can embed on their profile page.

2.2 Requirements analysis

In the REQUIREMENTS ANALYSIS DOCUMENT (APPENDIX D), both the functional as non-functional requirements of the project are specified. Most important is the feature list of the product. After a brainstorm, user stories have been defined for all features applicable to the application. These user stories have been categorized using the MoSCoW method as follows:

2.2.1 Must haves

- Coupling of Hyves accounts to messenger accounts (ACCOUNTS)
- Loading Hyves friends in Messenger (CONTACTS)
- Notifications of activity on Hyves via alerts (ALERTS)
- Access to Hyves user profiles from Messenger (PROFILES)

2.2.2 Should haves

- Mechanism to check for updates (UPDATES)
- Showing status of Hyves friends (STATUS)
- Enabling chat interoperability between Hyves friends and the Messenger client (CHAT)
- Show Hyves WhoWhatWhere (WWW)

2.2.3 Could haves

- Update Hyves WhoWhatWhere when changing PSM (WWWPSM)
- Show Hyves display pictures (AVATAR)
- Posting scraps to Hyves users (SCRAPS)
- Posting “tikken” to Hyves users (TIKKEN)
- Access to a mini-Hyves website (MINI)

2.2.4 Would haves

- Match Hyves contacts with Windows Live buddies and show them as a single user (MATCH)
- “Smart messenger groups” based on profile information in Hyves (GROUPS)
- Show latest blog posts and photos of Hyves users (ACTIVITY)
- Add support for all XMPP networks (XMPP)
- Integration with other social networks

This feature list has been used as a guideline throughout the entire project. The development iterations have been planned based on the above list.

2.3 Design and implementation

Once the development guidelines had been established in the requirements analysis and the most important research studies had been conducted, the design and implementation phase was ready to embark. In the DESIGN AND IMPLEMENTATION (APPENDIX E) the overall application design and implementation specifics are being discussed.

2.3.1 Initial design

To develop an initial design of the application, we started out by analyzing the features as they had been composed in the requirements analysis. Based on the user stories, four key pillars have been selected: network traffic interception, UI integration, the Messenger API and the Hyves API. Nearly all features depend on one or more of these key building blocks.

The overall application architecture has been developed based on this feature dependency analysis. An architectural overview can be seen in Figure 2-1, which is further explained in APPENDIX E.

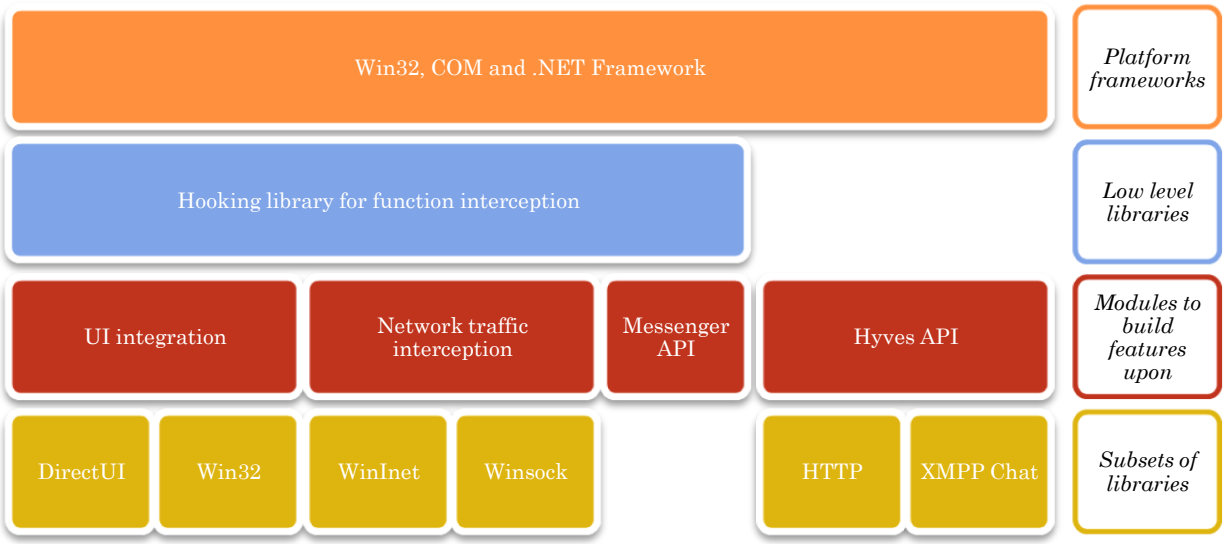

Figure 2-1: an architectural overview of the application

Based on the architectural design, the packages and their dependencies have been defined. Overall, the project contains four different packages / projects. Loader is the package responsible for loading our application when Windows Live Messenger starts. UIWrapper and Hooks are so-called mixed-mode assemblies written in a mix of managed and unmanaged C++ to expose native functionality to managed assemblies. UIWrapper functions as an interoperability layer managed assemblies can use to access Direct UI functions. The Hooks package exposes functionality for function interception to managed code. The Core package is where the main application logic resides. It is written completely in C# and combines functionality exposed by the UIWrapper and Hooks packages.

Because the Core package implements most of the application functionality, the design of this package has been explained more thoroughly. The Core design overview explains how the different tiers of the package communicate with each other. In the class overview the most important classes are being introduced and their relationships with each other can be seen in the class diagram (APPENDIX E).

2.3.2 Development iterations

After the overall application design had been agreed upon, the actual implementation phase was ready to start. According to the software development methodology chosen, in every development iteration a single user story would be completely implemented and tested. During the planning of the development iterations it was important to consider in what stage the building blocks (introduced above) required for every user story would be completed.

A development iteration in this project has been defined as developing and testing a complete user story together with the missing (parts of) building blocks the user story depends upon. This means development of the first user story requiring a particular building block would obviously take longer than when this building block has already been created during an earlier iteration.

2.3.3 Development techniques and tools

For the development of the application we have chosen to work with a toolset based upon Microsoft technology. Visual Studio 2008 has been used as integrated development environment (IDE), and all the packages are contained as projects within one solution. The Microsoft Detours library as introduced in the RESEARCH STUDY (APPENDIX B) is being used for function interception. C# has been chosen as the main development language, but C++ is being used for parts that could only not (easily) be implemented in managed code. The Microsoft Design Guidelines for Class Library Developers have been used as a guideline during the design and implementation phase.

Below is a list shortly describing other tools used during the development of the application:

- **Resharper (JetBrains)**
  Resharper is an add-on for Visual Studio which aids development in this IDE in various ways.
- **NUnit (NUnit.org)**
  NUnit is a xUnit based unit testing tool for .NET languages. The unit tests in the project use this tool.
- **Dependency walker (Microsoft)**
  Dependency Walker is a utility that scans any Windows modules and builds a hierarchical tree diagram of all dependent modules. For each module found, it lists all the functions that are exported by that module, and which of those functions are actually being called by other modules. This is particularly useful when looking which functions are used by a program so these can be intercepted.
- **Resource hacker (Angus Johnson)**
  Resource hacker is a tool with which resources can be extracted from windows executables. This has been useful to extract type libraries from Windows Live Messenger and to examine Direct UI related resource scripts.

- **HttpAnalyzer (IEInspector)**
  HttpAnalyzer does not only list all HTTP requests made on a system, it can also decrypt HTTPS traffic. This has been useful to see which HTTP requests would be useful by our application to intercept, and whether custom data had been injected correctly.
- **EtherDetect Packet Sniffer (EtherDetect)**
  A tool similar to HttpAnalyzer, but EtherDetect detects all network traffic, as opposed to only HTTP traffic. This tool has been useful to monitor MSNP data.
- **.NET Memory Profiler (SciTech Software)**
  .NET Memory Profiler is a tool for finding memory leaks and optimizing the memory usage in programs written in .NET. This tool gives a good insight in which objects are and which are not being garbage collected and for which reason.
- **Spy++ (Microsoft)**
  This tool can be used to monitor windows and window messages on the Windows operating system. It has mainly been used as a diagnostic tool to monitor window messages sent with information about asynchronous sockets.
- **Wrappit (Michael Chourdakis)**
  Wrappit is a tool to generate so-called proxy DLLs (APPENDIX C) dynamically. This has been used to create the Loader projects.

Throughout the entire project, subversion (SVN) has been used as a version control system. Besides all source code, also the related documents have been checked in to the repository.

2.3.4 Third party libraries used

Besides third party tools to aid in the development process, the final application also depends on three third party libraries, namely:

- **Detours (Microsoft Research)**
  The Detours library intercepts Win32 functions by re-writing the in-memory code of target functions (APPENDIX C).
- **BeeNET (Arian Geertsema)**
  Bee.NET is a Hyves development toolkit designed for .NET applications. It is being used by the application to communicate with the Hyves API.
- **jabber-net (open source, multiple contributors)**
  jabber-net is a set of .NET controls for sending and receiving Extensible Messaging and Presence Protocol (XMPP), also known as Jabber. It is being used by our application to connect to the Hyves chat servers.

2.4 Quality assurance and testing

This part explains what ways have been used to test the application code in order to assure the quality of the final product. So far, three different methods of testing have been used, namely unit testing, integration testing and alpha testing.

2.4.1 Unit testing

At the start of the project, the idea was to use unit testing for most of the classes. During the actual implementation phase however, we quickly found out this would be difficult to realize. The main difficulty is a substantial part of the application code relies on the behavior of Windows Live Messenger itself. The code that directly depends on the behavior of Messenger either successfully integrates with the application or it does not. In the latter case, this will be quickly noticed when running the application. Creating unit tests for these kinds of integration scenarios would be difficult because it is likely the testing code itself would also depend on the Messenger application. Other parts heavily integrating with third party libraries can therefore also be hard to write unit tests for.

Because of these conditions the effort to write a useful unit test for some scenarios would be out of proportion compared to the advantages. Therefore, it was decided to change the conditions in which to apply unit testing. The unit tests that have now been written mostly cover components that could be easily isolated from the rest of the application and Windows Live Messenger. Also, when a bug was encountered that was of a typical unit-testable nature (like an algorithmic error), unit tests have been written for the containing class before fixing the bug.

2.4.2 Integration testing

The advantage of using an iterative development process based on user stories is that the system is developed incrementally. After the completion of every user story a fully working application is available. This enabled us to use a working version of the application ever since the first user story had been completed. Whenever a development iteration was completed, the user story it was based on could immediately be tested for successful integration.

A particular advantage of continuously being able to run integration tests is the easiness to test the application on different platform configurations. As stated in the REQUIREMENTS ANALYSIS (APPENDIX D), it has been important for this project to be compatible with different versions of Windows Live Messenger. Integration testing aided us in this process. One developer, Jeroen, was continuously using version 8.5 of messenger, whereas Yousef had been using the version 9 beta. In this way we have made sure our extension is robust enough to support multiple versions of the underlying application.

2.4.3 Alpha testing

At the moment of writing, the application is also handed out to a small group of close friends of the developers. This will enable us to see how actual users (as opposed to developers) interact with the application. Also, it enables us to monitor the stability of the extension on even more platform configurations.

3 Conclusions

After finishing the phases related to the execution of the project as described above, what has been accomplished, what is there left to do, and how do I personally look back on the project? In this chapter these questions will be answered one-by-one.

3.1 Final application

At the time of writing, all must-have features have been completely implemented. All of the should-haves have also been implemented, except for an automatic update mechanism. Next to this, the Hyves display pictures of contacts are also being shown (could-have). This means we have been able to implement the most important functionality of the program.

One of the main non-functional requirements is to enable seamless integration of the extension with Windows Live Messenger. To see how this has been accomplished, a screenshot impression of the final application has been included as APPENDIX F.

3.2 Road ahead

Not only the most fundamental features have been implemented, but also all technical challenges encountered have been overcome. A comprehensive framework has been built to extend Windows Live Messenger, and it should now be relatively straightforward to build new features leveraging the existing codebase. This also means however, that the stability of the current version of the application will be key to the application's success.

To ensure this stability on different platform configurations, we plan to run a private beta test in December and January. About ten to thirty people should test the application during this period. This will give us time to work on a website presenting the product, add some additional features (which should be easy to test ourselves) and of course to fix issues encountered by testers.

Before starting the private beta it is essential to add enough tracing and exception handling code to ensure we can get a proper insight into bugs encountered. Also, when the application contains more features it will be necessary to write down scenarios (test cards) for acceptance tests, which until now have been executed without a storyboard.

After the private beta test a public or perhaps invitation-only beta version will be released on the website. During this period we also expect to be able to start incorporating support for other networks such as Google Talk and Facebook. Again, we do not expect to run into big problems while implementing features such as these once the stability of the underlying framework has been assured.

At this moment there are already several ideas for related products. One of them is to create a browser-based version of this project (similar to eBuddy or Meebo), allowing users to access their networks from any place at any time. Combined with this project (a desktop application) this can open up interesting scenarios such as cross-client settings / chat log synchronization. Another advantage of a browser-based application is that it will be easier to monetize using advertisements.

3.3 Reflection

This project has been different than any project I have done before in a couple of different ways.

First of all, the nature of the project has been relatively large and technically complicated. During the project, I have learned new things about a couple of different technologies such as mixed code, function interception and the Windows platform in general. I have been able to quickly incorporate these new technologies in my toolset and leverage them for the technical design of the project. The outcome of the project fulfills my expectations, although it has not been an easy job to extend Windows Live Messenger the way I have chosen too (on average I think I have spent about 45 – 55 hours a week on the project, resulting in about 70 pages of documentation and nearly 11000 lines of code). However, I expected this in advance and took this into account when choosing to work on this project.

The scope of the project for the first time made me experience the importance of a well thought through class design. Creating working prototypes is one thing, but incorporating them into production code in a scalable and understandable way is a completely different step, on which I have spent quite some time during this project. This has been a really positive experience for me and made me think more about the architectural difficulties of software design, and has sparked me to dive deeper into ideas behind different software design patterns.

Also, because this project functions as my bachelor graduation project, I have probably spent more time on documenting the different aspects of the project than I would have done otherwise. This has sometimes been difficult for me, because I generally prefer to spend this time thinking about new ideas or solving technical challenges. However, now most of the documentation has been completed, I can say writing and maintaining it has overall been a positive experience. I expect the documentation will be really helpful when other people have to get up to speed with the project, which has been the primary goal of writing it. The next time I work on a similar project, I suspect to write a similar document about the project plan, requirements and design and implementation as I have done in this project. It will depend on the nature of that project whether an extensive research study will be necessary as well, or whether a document with snippets from and references to other resources will satisfy.

Bibliography

(n.d.). Retrieved September 30, 2008, from MSNPiki: http://msnpiki.msnfanatic.com/

"kirin". (2005, September 26). *Hooking MSN Messenger 7 with VC++, A tutorial on how to hook MSN Messenger*. Retrieved September 30, 200, from Mess.be user forums: http://forum.mess.be/index.php?showtopic=11966

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis*. VTT Publications.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). Retrieved September 17, 2008, from Manifesto for Agile Software Development: http://www.agilemanifesto.org/

Boehm, B., & Turner, R. A. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley.

CACE Technologies. (n.d.). *WinPcap, The Packet Capture and Network Monitoring Library for Windows*. Retrieved October 2, 2008, from http://www.winpcap.org

Fitzpatrick, B., & Recordon, D. (2007, August 17). *Thoughts on the Social Graph*. Retrieved December 15, 2008, from http://www.bradfitz.com/social-graph-problem/

Hunt, G., & Brubacher, D. (1999). *Detours: Binary Interception of Win32 Functions*. Redmond: Microsoft Research.

Laganière, R., & Lethbridge, T. C. (2001). *Object-Orienented Software Engineering*. McGraw-Hill.

Levy, K. (n.d.). *Windows Live Messenger Add-ins beta feature for developers*. Retrieved September 28, 2008, from Ken Levy's Blog: http://blogs.msdn.com/klevy/archive/2006/05/10/Windows_Live_Messenger_Addins_beta_feature_for_developers.aspx

McConnel, S. (1996). *Rapid Development: Taming Wild Software Schedules*. Microsoft Press.

Microsoft Corporation. (2007, November). *Initalization of Mixed Assemblies*. Retrieved October 2, 2008, from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms173266.aspx

Microsoft Corporation. (n.d.). *Overview of the Windows Live Messenger Activity API*. Retrieved September 28, 2008, from http://msdn.microsoft.com/en-us/library/aa751014.aspx

Mintz, M. (n.d.). Retrieved September 30, 2008, from MSN Messenger Protocol: http://www.hypothetic.org/docs/msn/

NU.nl/Wieland van Dijk. (2007, August 1). *"Hyves is groter dan Google"*. Retrieved October 2, 2008, from nu.nl: http://www.nu.nl/news/1179455/50/%27Hyves_is_groter_dan_Google%27.html

Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development.* Prentice-Hall.

Wells, D. (1999). *The Rules and Practices of Extreme Programming*. Retrieved September 17, 2008, from Extreme Programming: http://www.extremeprogramming.org/rules.html

Wells, D. (1999). *What is Extreme Programming*. Retrieved September 17, 2008, from Extreme Programming: http://www.extremeprogramming.org/what.html

Wikipedia contributors. (2008, October 21). *Mediator pattern*. Retrieved October 27, 2008, from Wikipedia, the free encyclopedia: http://en.wikipedia.org/w/index.php?title=Mediator_pattern&oldid=246699775

Appendix A: Project description

(Dutch)

Voorstel bachelorproject Yousef El-Dardiry (9 september 2008)

Voor mijn bachelorproject wil ik één van mijn eigen ideeën uitwerken tot een daadwerkelijk project. Hiervoor zal ik alle stappen moeten doorlopen in het Software Design traject en contact moeten onderhouden met het bedrijfsleven, een contactpersoon binnen de TU Delft, en een programmeur van een andere universiteit. In dit document zal ik het project inhoudelijk beschrijven.

Inleiding

Het idee dat ik heb is om sociale netwerken (zoals Hyves en Facebook) te integreren in de desktop-ervaring van een gebruiker. In dit specifieke geval wil ik beginnen met het integreren van Hyves in Windows Live Messenger. Vanwege de grote overlappende gebruikersgroep zie ik dit als een goed startpunt. De opkomst van open standaarden, “mashups”, “social graph API’s” en discussies over “data portability” maken dit project tot een actueel idee in de internetwereld. Voordat ik in ga op de specifieke functies die ik voor ogen heb zal ik eerst een beschrijving geven van de twee betrokken diensten.

Betrokken diensten

Windows Live Messenger

Dit programma, voorheen bekend onder de naam “MSN Messenger”, is met meer dan 5,5 miljoen gebruikers veruit het meest gebruikte chatprogramma in Nederland.

Techniek: Windows Live Messenger maakt gebruik van een eigen chatprotocol (MSNP) om met de servers van Microsoft te communiceren. Er zijn API’s beschikbaar om contactgegevens op te halen, chatrobots te maken, en om kleine spelletjes / applicaties te maken die binnen het programma kunnen draaien.

Hyves

Hyves is in 2004 in Nederland opgericht en sindsdien uitgegroeid tot het meest populaire sociale netwerk van Nederland (ca. 6 miljoen gebruikers). Het contact maken en onderhouden van vriendschapsrelaties staat bij deze website centraal. Sinds kort kan men ook de status zien van Hyvers, en kunnen er gesprekken worden gestart (het zogenaamde kwekken) wanneer vrienden tegelijkertijd online zijn.

Techniek: Hyves stelt sinds kort applicaties van derden in staat met behulp van een API gegevens op te halen van haar gebruikers. Tevens kunnen er op profielen gadgets geplaatst worden die ontwikkeld zijn op het OpenSocial platform van Google. Het chatsysteem van Hyves is gebaseerd op de open standaard XMPP (voorheen Jabber) en daardoor gemakkelijk toegankelijk voor derde partijen.

Het combineren van functionaliteit

Omdat sociale netwerken en instant messaging clients dicht bij elkaar liggen (en steeds dichter naar elkaar toe groeien) is het denkbaar de de functionaliteit van de ene dienst geïntegreerd kan worden met die van de andere. Dit is precies wat ik voor ogen heb met mijn project; het integreren van het vriendennetwerk Hyves in het chatprogramma Windows Live Messenger. De volgende functies zou ik willen verwezenlijken:

- Het koppelen van een Hyves account aan een Windows Live ID, zodat wanneer men inlogt in Messenger, direct ook inlogt bij Hyves.
- Het laden van Hyves vrienden in de Windows Live Messenger contactlijst.
- Het reflecteren van de status van Hyves vrienden in Windows Live Messenger. En vice versa; het reflecteren van de Live Messenger status in Hyves.
- Het laden van schermafbeeldingen van Hyves, en associëren met de in Messenger geladen Hyves vrienden.
- Notificaties van activiteit op Hyves in de vorm van een *MSN Alert*.
- Het kunnen chatten met Hyves gebruikers vanuit Live Messenger.
- Het laden van profielinformatie van Hyves vrienden in Live Messenger.
- Het kunnen "krabbelen" (een berichtje op Hyves achterlaten) van Hyves vrienden vanuit Messenger.
- Et cetera. Er zijn nog talloze functies bedenkbaar die in een later stadium gecombineerd kunnen worden.

Dit alles moet worden gerealiseerd in de vorm van een uitbreiding op Live Messenger. Nadat de gebruiker deze uitbreiding heeft gedownload en geïnstalleerd, moet alle functionaliteit op een dusdanige manier integreren met Messenger zodat de gebruiker bij wijze van spreken nauwelijks merkt dat er een uitbreiding op de achtergrond meedraait.

Technische realisatie

Een grote uitdaging in dit project ligt bij het daadwerkelijk toevoegen van functionaliteit aan een bestaand programma, in dit geval Windows Live Messenger. Er zijn diverse API's beschikbaar om functionaliteit toe te voegen aan dit programma, maar geen van deze interfaces is toereikend genoeg om het niveau van integratie benodigd voor de bovenstaande functies te kunnen verwezenlijken. Hierdoor zal ik terug moeten grijpen naar een meer *low-level* aanpak.

Function detouring

Door middel van functie detouring is het mogelijk om functies die Messenger aanroept te onderscheppen, en te vervangen door eigen code. Door de juiste functies te onderscheppen moet het mogelijk zijn om aanpassingen te maken in de layout van het programma, en bijvoorbeeld de registratie van interne COM-interfaces te onderscheppen. Ik zou hiervoor een eigen detouring module kunnen schrijven, of gebruik kunnen maken van een bestaande library zoals *Detours* van Microsoft Research.

Netwerkverkeer onderscheppen

Met behulp van detouring is het ook mogelijk het netwerkverkeer van elke willekeurige applicatie te onderscheppen. Hierdoor kan de communicatie van Live Messenger met de

servers van Microsoft worden onderschept, en kan er op de juiste plekken eigen “hyves-data” worden ingevoegd. Om dit te realiseren zal ik hiervoor een eigen module moeten schrijven.

Uiteindelijk doel

Het einddoel van dit project is dat de hierboven beschreven applicatie verwezenlijkt wordt, en op internet wordt uitgebracht. Alle stadia die worden doorlopen zullen nauwkeurig gedocumenteerd moeten worden, zodat het project later zou kunnen worden overgedragen naar een derde partij.

Appendix B: Project plan

(Included separately)

Appendix C: Research study

(Included separately)

Appendix D: Requirements analysis

(Included separately)

Appendix E: Design and implementation

(Included separately)

Appendix F: Image walkthrough

Figure F-1: the sign in screen with the added option to select an associated Hyves account.

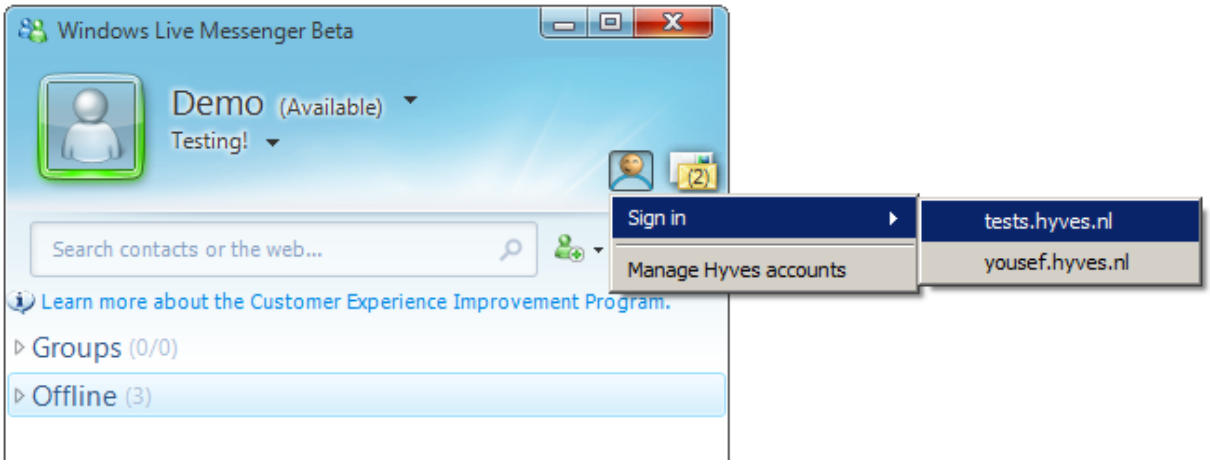

Figure F-2: the main window contains a Hyves button exposing several options of the extensions

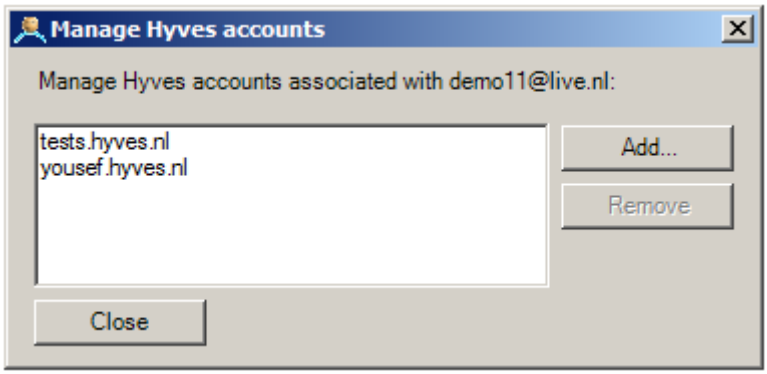

Figure F-3: the manage accounts dialog is at this moment the only dialog added by the extension. It allows the user to add and remove Hyves accounts associated with his Live ID.

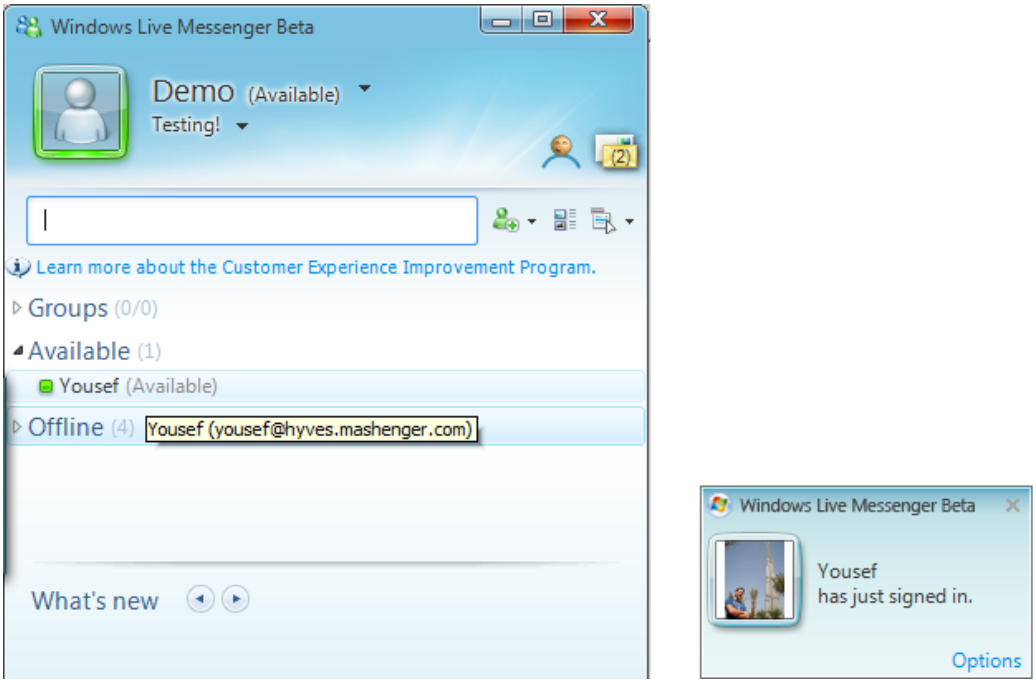

Figure F-4: After signing in with a Hyves account Hyves friends are added to the contact list. Statuses are reflected and notifications are shown in the same way as for regular Messenger contacts.

Figure F-5: chatting with online Hyves contacts works seamlessly. Neither the Messenger user (right window) nor the Hyves user (left / background window) notices he is talking to a user using another client. The Hyves picture of the contact is shown as Messenger display picture.

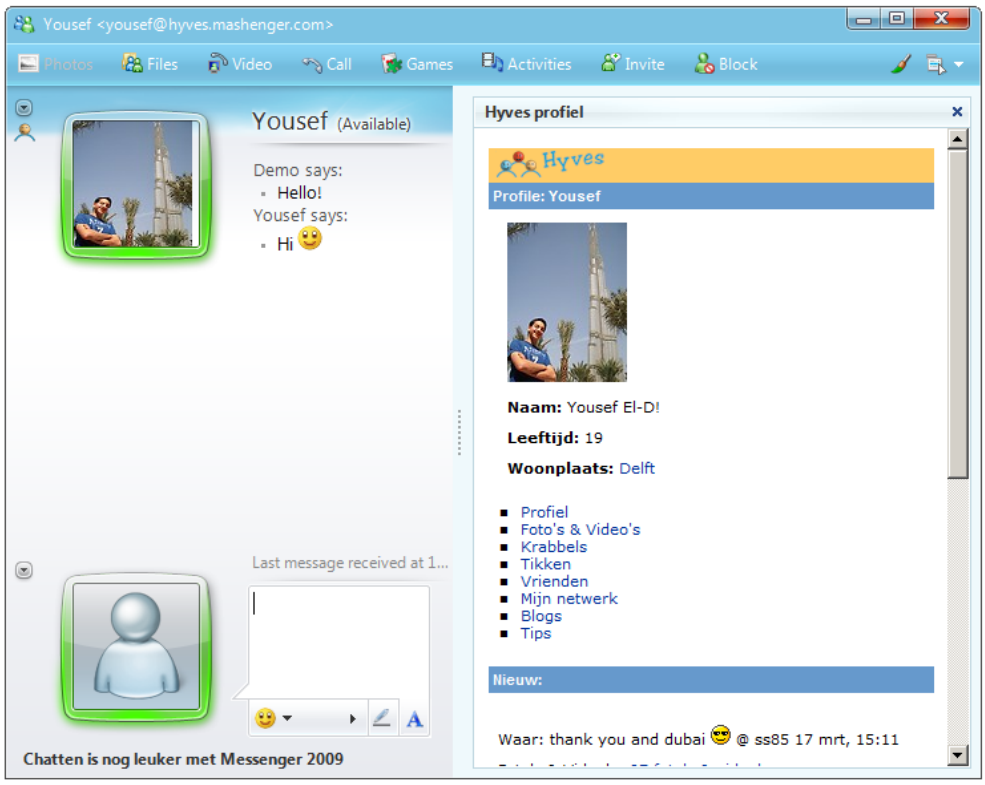

Figure F-6: a button is added to the conversation window indicating a Messenger to Hyves conversation (top left). Upon clicking the button, a Messenger activity is loaded showing the contact his Hyves profile.

9 september 2008

Projectomschrijving

Voorstel bachelorproject Yousef El-Dardiry

Voor mijn bachelorproject wil ik één van mijn eigen ideeën uitwerken tot een daadwerkelijk project. Hiervoor zal ik alle stappen moeten doorlopen in het Software Design traject en contact moeten onderhouden met het bedrijfsleven, een contactpersoon binnen de TU Delft, en een programmeur van een andere universiteit. In dit document zal ik het project inhoudelijk beschrijven.

Inleiding

Het idee dat ik heb is om sociale netwerken (zoals Hyves en Facebook) te integreren in de desktop-ervaring van een gebruiker. In dit specifieke geval wil ik beginnen met het integreren van Hyves in Windows Live Messenger. Vanwege de grote overlappende gebruikersgroep zie ik dit als een goed startpunt. De opkomst van open standaarden, “mashups”, “social graph API’s” en discussies over “data portability” maken dit project tot een actueel idee in de internetwereld. Voordat ik in ga op de specifieke functies die ik voor ogen heb zal ik eerst een beschrijving geven van de twee betrokken diensten.

Betrokken diensten

Windows Live Messenger

Dit programma, voorheen bekend onder de naam “MSN Messenger”, is met meer dan 5,5 miljoen gebruikers veruit het meest gebruikte chatprogramma in Nederland.

Techniek: Windows Live Messenger maakt gebruik van een eigen chatprotocol (MSNP) om met de servers van Microsoft te communiceren. Er zijn API’s beschikbaar om contactgegevens op te halen, chatrobots te maken, en om kleine spelletjes / applicaties te maken die binnen het programma kunnen draaien.

Hyves

Hyves is in 2004 in Nederland opgericht en sindsdien uitgegroeid tot het meest populaire sociale netwerk van Nederland (ca. 6 miljoen gebruikers). Het contact maken en onderhouden van vriendschapsrelaties staat bij deze website centraal. Sinds kort kan men ook de status zien van Hyvers, en kunnen er gesprekken worden gestart (het zogenaamde kwekken) wanneer vrienden tegelijkertijd online zijn.

Techniek: Hyves stelt sinds kort applicaties van derden in staat met behulp van een API gegevens op te halen van haar gebruikers. Tevens kunnen er op profielen gadgets geplaatst worden die ontwikkeld zijn op het OpenSocial platform van Google. Het chatsysteem van Hyves is gebaseerd op de open standaard XMPP (voorheen Jabber) en daardoor gemakkelijk toegankelijk voor derde partijen.

Het combineren van functionaliteit

Omdat sociale netwerken en instant messaging clients dicht bij elkaar liggen (en steeds dichter naar elkaar toe groeien) is het denkbaar de de functionaliteit van de ene dienst geïntegreerd kan worden met die van de andere. Dit is precies wat ik voor ogen heb met mijn project; het integreren van het vriendennetwerk Hyves in het chatprogramma Windows Live Messenger. De volgende functies zou ik willen verwezenlijken:

- Het koppelen van een Hyves account aan een Windows Live ID, zodat wanneer men inlogt in Messenger, direct ook inlogt bij Hyves.
- Het laden van Hyves vrienden in de Windows Live Messenger contactlijst.
- Het reflecteren van de status van Hyves vrienden in Windows Live Messenger. En vice versa; het reflecteren van de Live Messenger status in Hyves.
- Het laden van schermafbeeldingen van Hyves, en associëren met de in Messenger geladen Hyves vrienden.
- Notificaties van activiteit op Hyves in de vorm van een *MSN Alert*.
- Het kunnen chatten met Hyves gebruikers vanuit Live Messenger.
- Het laden van profielinformatie van Hyves vrienden in Live Messenger.
- Het kunnen "krabbelen" (een berichtje op Hyves achterlaten) van Hyves vrienden vanuit Messenger.
- Et cetera. Er zijn nog talloze functies bedenkbaar die in een later stadium gecombineerd kunnen worden.

Dit alles moet worden gerealiseerd in de vorm van een uitbreiding op Live Messenger. Nadat de gebruiker deze uitbreiding heeft gedownload en geïnstalleerd, moet alle functionaliteit op een dusdanige manier integreren met Messenger zodat de gebruiker bij wijze van spreken nauwelijks merkt dat er een uitbreiding op de achtergrond meedraait.

Technische realisatie

Een grote uitdaging in dit project ligt bij het daadwerkelijk toevoegen van functionaliteit aan een bestaand programma, in dit geval Windows Live Messenger. Er zijn diverse API's beschikbaar om functionaliteit toe te voegen aan dit programma, maar geen van deze interfaces is toereikend genoeg om het niveau van integratie benodigd voor de bovenstaande functies te kunnen verwezenlijken. Hierdoor zal ik terug moeten grijpen naar een meer *low-level* aanpak.

Function detouring

Door middel van functie detouring is het mogelijk om functies die Messenger aanroept te onderscheppen, en te vervangen door eigen code. Door de juiste functies te onderscheppen moet het mogelijk zijn om aanpassingen te maken in de layout van het programma, en bijvoorbeeld de registratie van interne COM-interfaces te onderscheppen. Ik zou hiervoor een eigen detouring module kunnen schrijven, of gebruik kunnen maken van een bestaande library zoals *Detours* van Microsoft Research.

Netwerkverkeer onderscheppen

Met behulp van detouring is het ook mogelijk het netwerkverkeer van elke willekeurige applicatie te onderscheppen. Hierdoor kan de communicatie van Live Messenger met de

servers van Microsoft worden onderschept, en kan er op de juiste plekken eigen “hyves-data” worden ingevoegd. Om dit te realiseren zal ik hiervoor een eigen module moeten schrijven.

Uiteindelijk doel

Het einddoel van dit project is dat de hierboven beschreven applicatie verwezenlijkt wordt, en op internet wordt uitgebracht. Alle stadia die worden doorlopen zullen nauwkeurig gedocumenteerd moeten worden, zodat het project later zou kunnen worden overgedragen naar een derde partij.

DEPARTMENT OF SOFTWARE ENGINEERING (EEMCS)
DELFT UNIVERSITY OF TECHNOLOGY

B. Project plan

BuddyFuse

Yousef El-Dardiry (1332686)

9/29/2008

Project plan

The project plan describes the basic setup of the project. The main project phases will be discussed, guidelines will be set on tools to be used and the people involved in the project will be introduced. Finally, the project plan contains a planning which will be adjusted during the course of the project.

Project Phases

The project consists of a number of different phases. In the research phase, a number of research questions will be answered to ensure there is enough knowledge on these areas to continue with the other phases.

Before we can start with the design and implementation of the project, it is important to have all requirements documented. This is the goal of the requirements phase.

Based on the software development method chosen in the research phase, the design and implementation phase will embark. In this phase the application will be designed, built and tested. Let us take a closer look at the three main phases.

Research phase

As previously stated, in this phase a couple of questions will be answered to establish a proper base for the rest of the project. The following three questions should be answered in a research study of about 3-5 pages per question.

- Which software design method is most applicable for use in this project?
- What are the technologies behind Messenger and Hyves, and what kind of APIs do these services expose?
- How can one add functionalities to existing compiled programs beyond the possibilities of exposed extensibility-APIs and without access to its source code?

Requirements phase

In the requirements phase all up-front requirements will be gathered. The final deliverable of this phase is a requirements analysis document including a project description and a so-called MoSCoW document listing the features by priority. Depending on the software development method chosen, other prerequisites for the next phase can be included in the requirements analysis document.

Design, implementation, and testing phases

In this phase the actual application will be developed and tested. The end product of this phase is a cleanly designed, working and thoroughly tested application fulfilling the requirements analyzed in the previous phase. Although the software development methodology to use is still to be chosen in the research study, due to the size of the project and team it is likely an iterative way of designing and developing will be used. The planning accompanying this document is based on this idea. Each development iteration should be summarized in the report, and if necessary technologic difficulties must be documented.

Tools and guidelines

It is important to set some guidelines on the tools and standards to be used in the project, so the team members will work with a toolset familiar to everyone involved.

Documentation

The documentation produced as a result of the different phases will be composed using Microsoft Word 2007. The layout of all documents must be consistent, at this moment the theme "Oriel" is in use. All documentation must be written in English.

Documents will also be checked in to source control in .docx format as a way to easy back up important files. Documents will be exported to PDF for reviewing.

Development

A list of tools to be used during the implementation and testing phase:

- **Source control system:** Subversion. All developers will have full access.
- **Preferred programming language:** The preferred language of both developers involved in the project is C#. When a lower level language is required we can fall back to C++.
- **Programming standards:** Microsoft .NET Design Guidelines for Class Library Developers.
- **Development tools:** Microsoft Visual Studio Professional 2008.
- **Testing tools:** NUnit.
- **Network analysis tools:** HttpAnalyzer and Etherdetect, or any similar tool.
- **Other tools that might become useful:** Dependency walker, Spy++, Resource Hacker.

Testing

Although the exact method of testing will be specified later on, it will be useful to discuss a couple of testing guidelines already. As mentioned above we will use the NUnit testing framework for unit testing. Next to unit testing, we plan on releasing a beta test of the software towards the end of the project. This way we can make sure the software functions properly on different computer configurations.

Deliverables

At the end of the project, the following documents should be contained in the final report:

- Project plan
- Research document
- Requirements analysis document, including MoSCoW and project description
- Design overview
- Reports of development iterations
- Review
- Code of final application

Since the intention of the project is to create a commercially viable product, the deliverables should give the impression they were created during a real-world commercial project. However, this does not impose academic aspects of the project should be neglected, but for example overhead in documentation should be kept at a minimum level. The intent of the report is to function as a reference for people involved in the project, and in the future for new team members to be able to get up to speed with the project quickly.

People involved and availability

Yousef El-Dardiry

As the project leader, I will be overall responsible for the project. It will be my task to assure all phases will be finished successfully. Most of the documentation, design and development will be done by me. Since the project is also my own idea, I will also function as main customer of the project. I will be available almost full-time until the end of the year.

Jeroen Bransen

Jeroen is a master student Cognitive Artificial Intelligence at the Utrecht University. He will assist in the development phase of the project. Since he does not play an active role in writing the report, he will also function as a mentor supervising the documents I write. Jeroen will be available about 3-4 days a week as a programmer until the end of the year.

Ir. B.R. Sodoyer

As the coordinator of bachelor projects at the Department of Software Technology Mr. Sodoyer will be my primary contact person at the Delft University of Technology. I intend to schedule a meeting with him at least once every two weeks to discuss the project progress. Every document written should be reviewed by and discussed with Mr. Sodoyer.

Planning

A rough planning is listed below. It will be updated with more details as the project progresses.

Week 1, 15/9 – 19/9
- First version of project plan

Week 2, 22/9 – 23/9 (abroad 24/9 – 27/9)
- Final version of project plan

Week 3, 29/9 – 3/10
- End of research phase

Week 4, 6/10 – 10/10
- First version of requirements document
- Initial design phase

Week 5, 13/10 – 17/10
- Final version of requirements document
- Start development iterations (Loader, Hooks & UIWrapper)

Week 6, 20/10 – 24/10
- Loader, Hooks & UIWrapper
- Development iterations (User stories 1-3)

Week 7, 27/10 – 1/11
- Development iterations (User stories 1-3)
- *22/10:* hand in documents for checkpoint review

Week 8, 3/11 – 7/11
- Development iterations (Finish stories 1-3, user stories 4, 6, 8)
- Checkpoint review

Week 9, 10/11 – 14/11 – abroad
- Development iterations (Finish user stories 4, 6, 8)

Week 10, 17/11 – 21/11
- Development iterations (User story 7)
- Draft overview document

Week 11, 24/11 – 28/11
- Development iterations (Finish user stories 1-8)
- Acceptance testing
- Code walkthrough, review overview document draft

Week 12, 1/12 – 5/12
- Development iterations (User stories 10)
- Acceptance testing

Week 13, 8/12 – 12/12
- System testing
- End development iterations (User stories 10)**,** review all code
- Hand in final code and documentation

Week 14, 15/12 – 19/12

- 17/12: Review with Jeroen and Dhr. Sodoyer

January 27: Presentation

DEPARTMENT OF SOFTWARE ENGINEERING (EEMCS)
DELFT UNIVERSITY OF TECHNOLOGY

C. Research study

BuddyFuse

Yousef El-Dardiry (1332686)

12/15/2008

Table of Contents

Introduction

The goal of this research study is to answer three questions important to the project. The results of this study will be used during the development of the application. For the design and development phase, it is important to know exactly what software development methodology we will follow during the course of the project. In part one, three different methodologies will be studied. Afterwards the decision will be made on what methodology to use in this project.

For the development of the program, we need to be able to deeply integrate our program with an existing application, Windows Live Messenger. How exactly can one deeply integrate into an existing application without access to its source code? To answer this question, function interception will be studied in part two.

Part three will take a look at the Hyves website and Windows Live Messenger application. Since we are planning to integrate these two services, it is important to know in what way we can connect with them. We will take a look at what APIs these services expose, and what techniques they are based upon we might be able to integrate with.

Part 1: Software development model

Which software development methodology is most applicable for use in this project?

Before we can start the design phase of the project, it is important to choose the most suitable software development method for the project. In this part we will take a number of different methods into consideration and compare them with each other. Then the nature of the project will be taken into consideration to find out what model is most applicable.

1 Introduction of models

Three different software development models will be compared in this chapter. Different software development models describe different perspectives of the software development process. Some focus on describing team coordination, whereas others focus on the design and development phases. Since this project only has a limited timeframe (three months) and a small team, three models will be compared focusing on the design and implementation phases.

The traditional waterfall model and two models in the *agile programming* family will be taken into consideration, namely extreme programming and feature driven development. Each of the models will be introduced with a short description, and their pros and cons will be listed.

1.1 Waterfall model

The waterfall model is considered to be the classic development methodology for software engineering. It is a model where one transitions from one phase to the next in a sequential way (Figure 1-1).

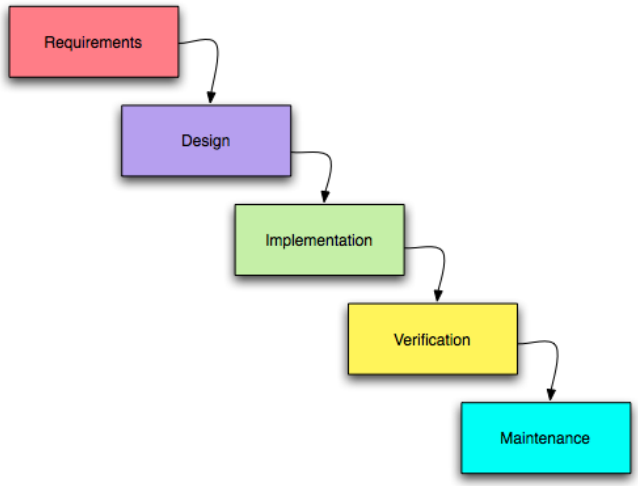

Figure 1-1, the waterfall model

The waterfall model requires the completion of a detailed documentation during the requirements and design phases before the implementation can start. In this way it tries to minimize the chance of running into any surprises at a later stage. However, it does recognize that sometimes one has to step back to an earlier stage when you encounter problems in a later stage (Laganière & Lethbridge, 2001).

Arguments for

A well thought through documentation process ensures that as much problems as possible will be identified up front. According to (McConnel, 1996): *"a requirements defect that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time."*. This means that the time you spend on the documentation up front will be returned since you will not run into as much surprises later on, when repairing these problems takes a lot of effort time.

Another argument in favor of the detailed documentation process of the waterfall model is that when the team changes during the project, it should be easy for new team members to get up to speed with the project (just by examining the documentation).

Arguments against

(Laganière & Lethbridge, 2001) have mentioned two important disadvantages of the waterfall model.

First, when following the waterfall model one has to complete the requirements and specifications before the design starts and the design before the implementation. However, this means that all requirements should be known up front, and cannot change during the project.

The second problem with the model is that it assumes one can get the requirements and design right simply by writing them down. Many software projects are not of such a simple nature to make this possible. These projects require a sort of trials such as prototyping to help in gathering requirements from the users and to find out what the proper use of the development tools and technologies is.

1.2 Agile methods

In contrast to sequential development methods such as the waterfall model, agile software development methodologies promote software development in an iterative way. The main ideas behind agile software development are stated in the Manifesto for Agile Software Development (Beck, et al., 2001):

"We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more."

While the basis of the waterfall model consists of comprehensive documentation and a detailed plan, agile development methodologies tend to be more lightweight. However,

this does not mean that the agile projects follow the so-called opportunistic approach of software development (modifying an initial prototype until satisfied).

Different agile methodologies have been developed to ensure agile projects follow a specified path to realize the final product. In this section two of these agile methodologies will be examined, namely extreme programming (XP) and feature driven development (FDD).

Agile methodologies tend to be useful for projects of a different nature than plan-driven methodologies such as the waterfall model. According to (Boehm & Turner, 2004) both types of software development have their own home ground:

Agile	Plan-driven
• Low criticality	• High criticality
• Senior developers	• Junior developers
• Requirements change very often	• Requirements don't change too often
• Small number of developers	• Large number of developers
• Culture that thrives on chaos	• Culture that demands order

1.2.1 Extreme Programming (XP)

Extreme programming tries to improve software development in four essential aspects: communication, simplicity, feedback and courage (Wells, What is Extreme Programming, 1999). Extreme Programming favors frequent communication about project requirements over extensive documentation. Simplicity in the project is established by starting with building the simplest solution. Extra functionality can be implemented later on, and should not be taken into account during early design and development. Feedback is gathered by writing unit tests early on and also by delivering a first version of the application to the customer as early as possible. Developers following the extreme programming methodology should be able to react to changing requirements and technology courageously, and they must know when to refactor or throw away existing code.

So-called user stories are created by the customer and are used as a guideline in release and iteration planning. A user story is a description of a feature in a couple of lines, not containing any technical terms. Programming tasks are created out of the user stories and assigned to different programmers.

A set of rules and practices for extreme programming have been composed and are divided into four main categories; planning, designing, coding and testing (Wells, The Rules and Practices of Extreme Programming, 1999). However, the method acknowledges not all rules might be useful for every project and states rules should be changed if they turn out not to work properly during the course of a particular project.

Arguments for

Arguments in favor of extreme programming tend to be quite straightforward. The product delivered at the end of the extreme programming trajectory has been thoroughly tested and the code should be as simple as possible. The customer is involved during the entire development process, so it should be easier to achieve a high level of customer

satisfaction. Being one of the agile software methodologies, projects following the extreme programming method can also quickly respond to changing requirements.

Arguments against

The lack of comprehensive documentation and design might not be suitable for large or mission critical projects. It also makes it harder for new team members to get up to speed with the project.

Another important criticism on extreme programming is the lack of detailed guidelines. Methods used in extreme programming are only briefly described and it acknowledges not all practices have to be used in the same project. This might result in practitioners to only adopt the rules they like, instead of adopting the rules the project would favor from most.

1.2.2 Feature Driven Development (FDD)

Feature driven development focuses on the design and implementation phases of a project. It differentiates five such phases (Figure 1-2, Feature Driven Development). In the first phase, an overall model of the application design is being developed. Based on a high-level description of the system (called a walkthrough), the system is divided into a number of sub domains. Object models for the different domains are designed, as well as an overall model for the system.

In the next step a comprehensive list of customer-valued features is composed. Features are divided in different sets and they must be reviewed by the customer to ensure completeness. During the plan by feature phase the features are ranked according to their priority and dependencies. Every feature will be assigned to a programmer (or team).

During the next two iterative processes, every feature is designed, build and tested one at a time (or in parallel when different people can work on a different feature). Programmers do not start working on a next feature unless the feature they have been working on has been completely tested. Every iteration takes from a couple of days to a maximum of two weeks.

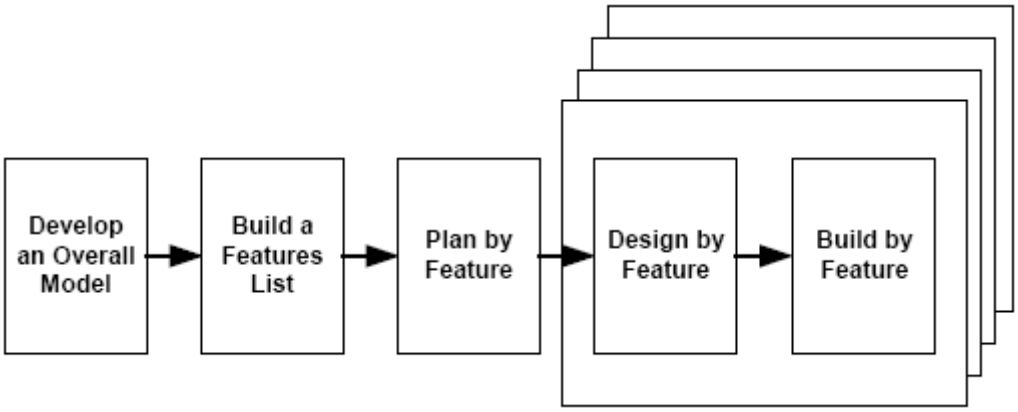

Figure 1-2, Feature Driven Development (Palmer & Felsing, 2002)

Arguments for

Similar to most agile programming methods, FDD delivers a completely working and tested program at the end of every iteration. In this way customers can stay involved with the project by reviewing the product during development.

In contrast to most other agile methods, feature driven development also claims to be suitable for large and critical systems (Palmer & Felsing, 2002). The two design phases (the first phase of the project, and the first phase of every development iteration) make sure a lot of design is done up-front, which makes FDD one of the more planned agile methods.

Arguments against

These comprehensive design phases can also be a source of critique on FDD, since it might not be as agile as methods like extreme programming. Another argument against FDD is that all the design happening up front is for code to be implemented within at most a couple of months, but design flaws might take longer to surface.

2 Comparison

A comparison between the different methodologies can be made based upon the above descriptions. The methodologies will be compared by their ability to respond to changing requirements, ability to respond to changes in the development team, their suitability for small teams and short projects and suitability for critical projects.

Methodology	Response to changing requirements	Ability to adopt new team members	Suitability for small teams	Suitability for short projects	Suitability for critical projects
Waterfall	--	++	-	-	++
FDD	+	+	+	+	+
XP	++	-	++	++	--

Table 1, a comparison of different software methodologies (-- very bad, - bad, + good, ++ very good)

The results of our study above have been summarized in Table 1. The waterfall method tends to be more suitable for mission critical projects, and is able to support changes in the development team because of its extensive documentation. However, this same documentation makes it not very suitable for small teams and short projects, and makes it hard to quickly adopt changing requirements.

The Extreme Programming practice in contrast handles changing requirements very well, and has been developed to be used by small teams. Its ability to adopt new team members has been based on these small team sizes. It is not sufficient enough for a new developer to read through the documentation to get up to speed with the project. As stated above, XP also is not suitable for very large, critical projects.

Feature Driven Development tends to be in the middle of XP and the waterfall model. Its documentation is more comprehensive than in XP, and causes the method to be suitable for larger and mission critical projects as well. The same documentation also means it is able to respond more effectively to changes in the project team formation.

Being an iterative agile development methodology, it is also able to respond to changing requirements and suitable for small teams and projects.

3 Software development method applicable

Now we have learned about different software development methodologies, it is time to reflect on their positive and negative values combined with the nature of the project to be developed.

3.1 Project nature

As described in the PROJECT PLAN, the project has a couple of constraints. The programming work will be done by only two people and it should be finished within three months. The lead developer also takes on the role as customer and project manager.

The PROJECT DESCRIPTION also puts us in front of a number of challenges. Integrating the software product into a third party application (Windows Live Messenger) might become a technical challenge requiring deep Windows programming knowledge and quite some experimentation up front. Most of the features in the application will have to be built based on these experiments. New releases of Windows Live Messenger or the Hyves API might affect the requirements during the course of the project.

3.2 Conclusion

The project nature described above has a couple of similarities with the agile home ground introduced in paragraph 1.2 above. A small team will execute the project and requirements can change during development. Although the programmers on the project might not be considered as senior programmers, both of them are expert in application development on top of the Windows Live platform and in this way they will bring a lot of experience.

The waterfall model does not seem to be suitable to be used because of the possibly changing requirements. Next to that, it will be hard to plan for technically difficult areas in the design that might require prototyping up front.

This means one of the agile methodologies might be more suitable for this project. The biggest differences between extreme programming and feature driven development are the more comprehensive design phases in the latter. Since most of the features require deep integration with the Messenger application, it seems to be a good idea to build a basic framework exposing the kind of integration needed. It makes sense to design such a framework up front, so features can be built on top of the framework more effectively later on. In contrast to FDD's design phases extreme programming's "simplest implementation first" approach does not seem very suitable for this part.

However, FDD has also been designed to be used in large teams and projects. Since we are dealing with a small project and a small team, some ideas of extreme programming might also be useful. For example, because the role of project leader will be fulfilled by the same person who will implement most of the program, it seems reasonable for him to make adjustments to the code on the fly by refactoring (like promoted in XP). Thus, it might be useful to combine some of the ideas of FDD with extreme programming. This seems feasible to do because the user stories based approach

of extreme programming has a lot of similarities with the feature based approach described in FDD.

Based on the above discussion, we propose to follow the extreme programming methodology with a couple of adjustments. The tasks composed of the user stories should be divided into different domains, similar as how features are categorized in FDD. Before starting on a task belonging to a particular domain, that domain should be properly designed and an underlying framework required by all the tasks belonging to the domain should have been built first. The global design of the different domains should be well documented. Just like in feature driven development, a task should be completely tested before moving on to the next task.

Part 2: Integrating with existing applications

How can one add functionalities to existing compiled programs beyond the possibilities of exposed extensibility-APIs and without access to its source code?

Without the source code of an existing application / program, it is a cumbersome task to add new functionality to it. For our project we will need to extend Windows Live Messenger with new buttons, contacts and other functionality. In this study we will take a look at a method to accomplish this called function interception. It should be mentioned other methods exist to modify an application's functionality. For example, binary patching of executables is often used to make small adjustments to a program. Sometimes, the operating system provides methods to "hook" into a running application; this method is used by for example computer based training or accessibility applications.

We have chosen to focus on function interception since it seems the most suitable method for the level of integration required. Next to this, existing extensions for Windows Live Messenger based on function interception have already proven this method is suitable.

Because of the nature of our project we will also focus on programs written for the Win32 platform, but the ideas behind the described methods should be applicable to other platforms as well.

What do we actually mean by adding functionalities to an existing application or program? Normally, when one wants to add functionality to a program you modify its source code and add or modify its procedures. Different procedures get called during the execution of a program, and the execution order of these procedures defines the behavior of the program. This means that if we can make modifications to the execution of different procedures, we essentially can modify the functionality of a program.

A lot of these procedures are contained within the program. Because it is difficult to find out what the task of such an internal procedure is without the source code or its documentation, it is hard to modify calls to and from these procedures in a meaningful way. However, the majority of procedures called during the execution of a program reside in other libraries. These libraries might contain well documented functions, for example those exposed by the underlying operating system. At least they almost always have a meaningful name by which other programs can address them. We hope that by monitoring and modifying these function calls and their return values we can eventually change the functionality of the program. Let us take a look at different methods to accomplish intercepting these function calls. Afterwards, the techniques studied will be used to explain how we can monitor the network traffic in a target application.

1 DLL injection

Before we can explain techniques to accomplish function interception in the next section, it is important to introduce the concept of code injection. Two of the methods to intercept function calls described below, will require writing to the memory space of the target application. This cannot be achieved however, when the interception code does not reside in the address space of the target process. A number of different methods exist to force loading of a dynamic link library into another process on the Windows platform, a technique referred to as DLL injection.

- Register the DLL as an AppInit DLL in the system registry. DLLs listed here will be automatically loaded into every process that loads the operating system library User32.dll.
- Use the *SetWindowsHookEx* or *CreateRemoteThread* functions to force injection of a DLL into an already running process.
- Place a DLL with the same name as a library loaded by the target application in the directory of that application. This *proxy DLL* will now be loaded instead of the original one. Be sure to forward function calls to the original library, so the proxy DLL does not break the target application its functionality.

The disadvantage of the first method is the DLL to be injected will be loaded into every process loading User32.dll. This can waste memory of the computer if DLL injection was only necessary to inject into a single process. In a similar way the second method might be unsuitable if one wants to inject into a target process every time it is loaded, since one must monitor whenever it is running to initiate the hooking process. The third method is more suitable for this task, since the target application will load it automatically.

2 Intercepting function calls

Table 2 shows what happens exactly when an application makes a call to a function residing in a dynamically linked external library. It uses the Win32 function *SendMessageW* - which resides in the user32.dll library - as an example. When the application code calls the function, it does not do this directly, since there is no way for it to know where the function has been loaded in memory. Instead, the program changes execution to the entry for the imported function in the Import Address Table (IAT). This table contains entries for all functions residing in external libraries imported by the program. When the executable is being run, the dynamic linker fills in the slots in the IAT of every linked function with an unconditional jump to the address of the actual function code. Using this table we will explain three different methods of function interception, namely proxy libraries, modification of an application's Import Address Table (IAT) and modification of the target function.

Type	Address	Data	Description
Import Address Table	0x00000010	jmp 0x10000000	USER32.SendMessageW IAT entry
	…	…	
Application code	0x00000100	call 0x0000010	call USER32.SendMessageW
	…	…	
User32.dll code	0x10000000	push ebx	start of SendMessageW
	…	…	

Table 2, simplified view of a call to SendMessageW in USER32.dll

2.1 Proxy libraries

One way to intercept the intermodular call would be to write a dynamic link library similar to the one containing the function we are trying to intercept. The target application must then load this proxy DLL instead of the original one. This can be enforced by naming the proxy DLL the same as the original one and copying it to the application's directory, possibly overwriting the original version. To make sure the target application will still behave normally, the proxy DLL must expose the same functions as the original DLL, and forward all function calls to the original DLL. If we look at Table 2, this would mean that the address 0x10000000 would not contain the code of the original User32.dll file, but the code of the proxy DLL. This process is essentially the same as described above as a method of DLL injection.

Since all function calls to the target library will be rerouted via the proxy DLL, it is possible to change the way the call will be handled. The code in the proxy DLL can modify parameters, return values, or choose not to forward the call at all.

The advantage of this method is that it is relatively simple to accomplish. As soon as we have written the proxy DLL, the operating system DLL loader makes sure calls are forwarded to our DLL. There is no advanced technique involved such as modifying the application's memory. However, it requires us to write separate DLLs for every module imported, and these DLLs might be difficult to maintain.

2.2 Import Address Table modification

Another method to achieve function interception is to modify the target application's Import Address Table during runtime. In Table 2 this means overwriting the data at 0x00000010 with an unconditional jump to a replacement function. When the application now calls the target function, the replacement function will be called instead. The replacement function can then execute custom code, and if necessary call the original function before returning execution to the caller.

Function interception by this method does however not intercept every function call to the target method, but depends on how the function is invoked. Figure 2-1, displays a schematic view of how a function pointer can be obtained to a function (*TargetFunction*) in a dynamically loaded module (*TargetModule*) can be retrieved (and thus invoked).

With this figure it can be shown what function calls can be intercepted by modifying the Import Address Table of *Executable.exe*. When the entry of "TargetFunction" is modified, only the dashed pointer will point to a new function. This means only

invocations using this pointer will be intercepted. As can be seen, there are still two other pointers pointing to the original not intercepted function.

One of these pointers is located in the Import Address Table of another module (*Module.dll)* loaded by the process. When a function compiled in *Executable.exe* calls *ModuleFunction* residing in this other module, *ModuleFunction* function might call *TargetFunction* by looking up its address in the non-modified address table of *Module.dll*. This means indirect calls to *TargetFunction* are not intercepted, unless the Import Address Tables of all loaded modules are modified.

The other pointer comes from *Kernel32.dll*. This operating system library can be used to dynamically link a module at run-time. With the *LoadLibrary* function, a module can be loaded at run-time. Then when this library is loaded, function pointers of exported functions can be retrieved with the *GetProcAddress* function. In the figure, when a function in *Executable.exe* uses *GetProcAddress* to obtain a function pointer to *TargetFunction*, function calls to this pointer will not be intercepted. This can be overcome by modifying the IAT-entry of *GetProcAddress* itself to point to an interception function returning the address of the interception function for *TargetFunction* instead of the original value.

Depending on the goal of the interception function, it can be either a disadvantage or an advantage that not every function call to the target function is intercepted. For example, if function interception is used to instrument all calls originating from *Executable.exe* to *TargetFunction*, IAT-modification can play a useful role. However, if one wants to intercept each and every call from a specific process to *TargetFunction*, it might be a cumbersome task to accomplish using IAT-modification.

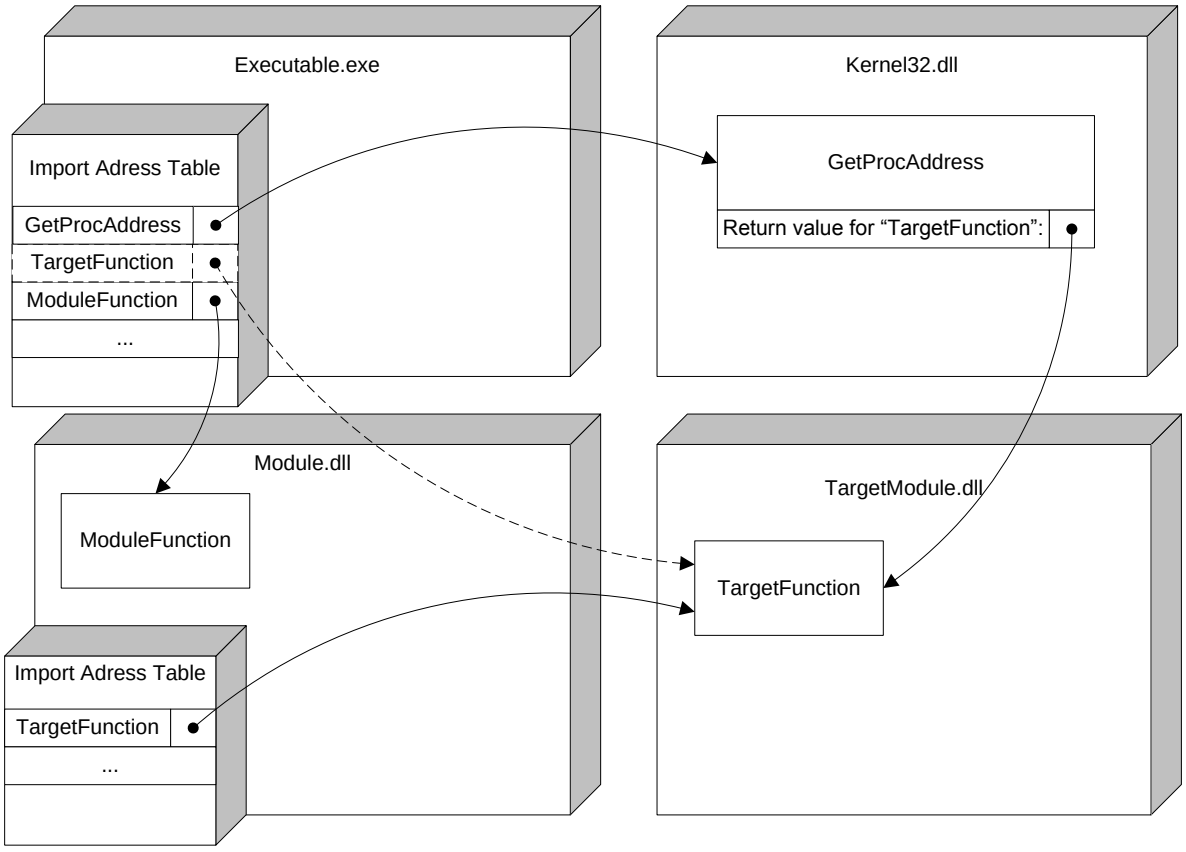

Figure 2-1, different ways a dynamically linked procedure can be invoked inside a single process

2.3 Modifying a target function in-memory

The third way we discuss to intercept functions involves modifying the data at the address of the target function. By overwriting the first couple of bytes of the target function with an unconditional jump to a replacement function, function calls to the target function will be intercepted. In Table 2, this would mean writing the binary code of an unconditional jump at memory address 0x10000000. The difficulty in this method comes when the replacement function needs to call the original function, since the first few bytes of the target function have been overwritten with a jump instruction. This means these bytes must be backed up, and executed before continuing execution at the remaining part of the intercepted function.

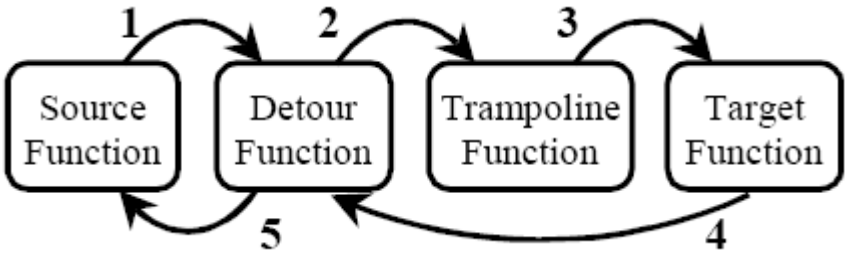

Figure 2-2, invocation of a detoured function (Hunt & Brubacher, 1999)

Detours is a library developed by Microsoft Research that accomplishes this using a so-called trampoline function. The trampoline function is a function that can be called by the replacement function (detour function) which executes the overwritten bytes and then calls the remainder of the original target function (Figure 2-2). The Detours library is available freely for non-commercial use. A disadvantage of using Detours is the licensing costs for commercial uses. The main advantage of modifying the actual target function is that it works regardless of the method the application uses to locate the target function. (Hunt & Brubacher, 1999)

3 Intercepting the network traffic of an application

We can now apply the techniques described above to intercept the network traffic of a target application. Let us see how we can intercept TCP and HTTP traffic of a target application.

3.1 Winsock

On the Windows platform, most applications use the Winsock libraries to create TCP and UDP sockets. A socket is created using the *socket* function. After creation, the socket can connect to a server using *connect*. When the connection has been established, the *send* and *recv* functions are used to respectively send and receive data packets. The last three functions will normally block the application execution until new information arrives from the network driver. However, the *WSAAsyncSelect* function can be used to enable asynchronous functionality. This will make the calls non-blocking, and a window-message will be posted to the application's window procedure when new information is available (for example, when a new packet arrives). The *WSAEventSelect* puts the socket into non-blocking mode in a similar way, but in this section we will not take this function into consideration for the sake of simplicity.

By intercepting function calls to *recv* and *send* (located in Ws2_32.dll), it is now possible to intercept incoming and outgoing data. The injected function can even modify the buffer before it is sent to the original function, or modify the incoming buffer before it is sent to the application. This process is illustrated in Figure 3-1.

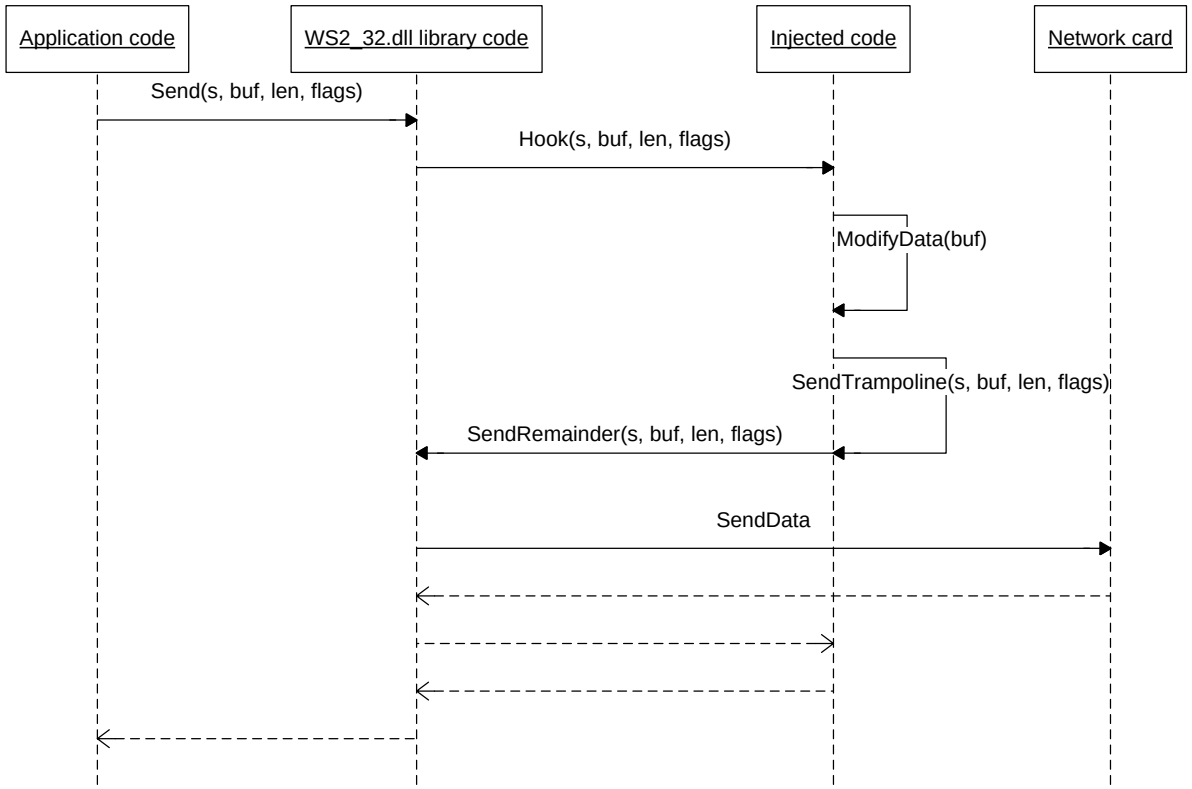

Figure 3-1, A schematic view of an injected process sending data using Winsock

If the application needs to inject its own data into the socket, it can do so by intercepting calls to *recv*, resulting in a similar execution flow as illustrated in Figure 3-1 for *send*. However, in case of an asynchronous socket it is likely the target application will only call the *recv* function when receiving a FD_READ message into its window procedure. This means it is not possible to inject receiving data an arbitrary moment, since the *recv* hook might not be called at that moment.

It is possible however to trigger the host application to call *recv*. This can be achieved by manually sending the FD_READ message to the target application. To find out to what window this message must be sent and with which parameters, it is possible to intercept the *WSAAsyncSelect* function. By doing so, one can also find out which sockets and which operations are selected for asynchronous behavior.

3.2 WinInet

Although Winsock hooking seems suitable to intercept most network traffic, sometimes it might be easier to apply a hook at a higher level. In this section we will introduce a different way to intercept HTTP requests made by a process.

Most applications on the Windows platform use the WinInet library provided by Microsoft Internet Explorer to make HTTP requests. The WinInet library exposes features to create HTTP, FTP and Gopher requests. The idea to intercept HTTP requests executed by the WinInet library is slightly different than the idea we of intercepting raw Winsock packets described above.

A similar method to intercepting calls to *recv* and *send*, would be to intercept WinInet functions *InternetReadFile* and *InternetWriteFile*. However, the target application might read and write requests part by part, synchronously or asynchronously, while most likely the injecting application needs to know details of all parts before it can forward the original function call.

A different approach would be to intercept calls made to *InternetConnect*. This function takes eight parameters, two of them being the target server name and port number. By hosting a (local) proxy server and changing the server name and port to the configuration of this proxy server, all calls will be rerouted to the proxy server. In this way the proxy server can take care of forwarding requests and responses, with the ability to modify sent and received.

With these two examples we have shown two different applications of function interception and explained this method can be useful to intercept both low and high level API calls. Note that there are other ways to intercept network traffic, for example by writing a low level driver, an approach taken by the widely used WinPcap library (CACE Technologies).

Part 3: Interaction with services

What are the technologies behind Messenger and Hyves, and what kind of APIs do these services expose?

Since the application we are about to develop is going to interact with two programs, it is important to give an introduction to these programs and the way we can interact with them. We will take a closer look at both Windows Live Messenger and Hyves, the technologies they are built upon, and the Application Programming Interfaces (APIs) they expose for third party applications.

1 Windows Live Messenger

Windows Live Messenger is an instant messaging application developed by Microsoft. Formerly known as MSN Messenger, it is currently the most popular instant messaging client in the Netherlands, and one of the most popular worldwide. Over the years, many features have been added to the application, including a different number of APIs. Let us take a closer look at the technologies used by the application itself, and then explore the different interfaces the application exposes for third party applications.

1.1 Technologies

Since Windows Live Messenger is closed software, it is a difficult task to acquire information about its inner workings. We will have to use a number of different third party tools to find out more about it, and leverage information on the internet other developers have already found out about the program.

By using a tool like *Dependency Walker*, one can quickly find out what libraries and functions are used by a specific executable on the Windows platform. *Spy++* is a tool that can supply us with information about the Win32 windows created by the messenger process, and messages sent to the different windows. *Resource Hacker* can expose the resources embedded inside Win32 PE files. Finally, we can use a tool like *Etherdetect* to monitor the network traffic of a computer, and thus find out how and what information the application communicates over the network.

1.1.1 Windows Live Platform

Windows Live Messenger is part of the Windows Live platform. It makes heavy use of services exposed on the Windows Live platform, and exposes its own services under this umbrella. This means it uses other Windows Live technologies to store different kinds of information. Windows Live Contacts is the platform used to store contact list information. Products such as Mail, Spaces and Alerts are integrated in the client.

1.1.2 Communication

The instant Messaging protocol Messenger uses is called MSNP. MSNP is not an open standard, so again we are facing the problem no official information is available about it. However, over the years a lot of research has been done by third parties on the protocol, and most of it is documented at websites such as (MSNPiki) and (Mintz).

Additional information about the protocol can be gathered by examining network traffic with a tool like Etherdetect.

1.1.3 UI Platform

When we use Spy++ to obtain information about the windows created by messenger, we see most of the window names contain "DirectUI". Next to this, Spy++ does not recognize any of the subwindows or buttons of the application. This means the controls drawn by messenger are so-called windowless controls. Messenger does not use a publicly available library such as Windows Forms or Windows Presentation Foundation to generate its user interface. Instead, it uses a library called DirectUI. DirectUI is an internal Microsoft product used in parts of Windows XP and Windows Vista, and not much information is publicly available about it. Using dependency walker, we can see that Messenger uses the libraries *ux*.dll* and *msncore.dll* (depending on the application version) to render its user interface.

By referencing these functions and intercepting the calls Messenger makes to them (described in Intercepting function calls above) it is hoped our application can make modifications to the Messenger user interface.

1.2 Application Programming Interfaces

Over the years Microsoft has released a different number of APIs for Windows Live Messenger and the Windows Live platform. Here is an overview of APIs that might become useful in the development of our application.

1.2.1 Activity API

The Windows Live Messenger Activity API allows developers to build applications that take advantage of the multiuser communication functionality provided by Windows Live Messenger. Developers can design applications using this simplified connection model.

The application in the activity window is a web page that interacts with Windows Live Messenger through the application programming interface that is defined in the software development kit (SDK). The web page can provide any functionality that can be present in a normal Web page. The web page is hosted in a messenger conversation window. (Microsoft Corporation)

1.2.2 Windows Live Agents

A Windows Live Agent is an interactive robot that analyzes an end-user's queries and matches them to the topic questions that are stored in the knowledge database. Windows Live Agents can run on the Messenger network. In this way, an Agent "simulates" a regular instant messaging contact.

1.2.3 Windows Messenger API

A different version of Windows Live Messenger, called Windows Messenger, exposes an advanced COM-based API allowing third parties to interact with the core components of Messenger. Although support for these APIs has been discontinued, the COM Type Libraries are still embedded in the latest versions of Windows Live Messenger.

Using resource hacker, we have been able to extract all type libraries embedded in the main Windows Live Messenger executable msnmsgr.exe. Using the Visual Studio

Object Browser tool, we can explore the types contained in the library. It turns out Windows Live Messenger still uses these APIs internally. As described by ("kirin", 2005), an application can take hold of these interfaces by intercepting calls to the Component Object Model (COM) CoRegisterClassObject function.

1.2.4 Add-in SDK

Similar to the Windows Messenger API described in 1.2.3 above, Microsoft also released a tryout version of an Add-In SDK in version 8 of Windows Live Messenger (Levy). However, support for this extensibility model has been discontinued as well.

1.2.5 Windows Live Contacts API

Using the Windows Live Contacts API, developers can use the functionality of the Windows Live Contacts address book service. Since Windows Live Messenger loads its address book from this service, changes made to the Windows Live Contacts address book will also be reflected in the messenger client.

1.2.6 Windows Live Alerts

Windows Live Alerts is a service that can be used by third parties to deliver notifications to subscribed Windows Live users. Windows Live Messenger is one of the platforms where alerts will be delivered, in the form of so-called toast-popups.

2 Hyves

Hyves is a social networking tool similar to MySpace or Facebook. It is primarily used in the Netherlands, where it has grown to one of the country's most popular web sites. (NU.nl/Wieland van Dijk, 2007)

Hyves is a web application and does not run on the desktop of the end user. This means the technology used in their platform will mostly be irrelevant for us, since we will not be able to integrate with these technologies directly. Besides screen-scraping the website which is not approved by Hyves, we can only interact with the APIs exposed by the service, so let us take a closer look at these services.

2.1 Application Programming Interfaces

After the success of the Facebook application platform released in 2007, other social networks around the world followed its example. In this section we will take a look at three different ways Hyves currently allows us to interact with its service, being with their web based API, Chat platform and gadget platform.

2.1.1 Hyves API

The Hyves API is a request based interface to retrieve and submit information from and to the Hyves platform. Using the OAuth authorization protocol, third party applications can make requests on behalf of the end user. Information programs can interact with using the Hyves API are photos, blog posts, friends, gadgets, hangouts, tips, scraps, and WWWs ("WhoWhatWhere" information).

Another feature of the Hyves API is to request an authentication token for the Hyves Chat platform described below.

2.1.2 Chat platform

Using a Hyves chat authentication token the application can sign in to the Hyves chat service on the end user's behalf. The chat server is based on the Extensible Messaging and Presence Protocol (XMPP, formerly known as jabber), an open standard for real time communication.

Using the chat platform applications can send instant messages to Hyves friends. The XMPP server also notifies connected applications of the presence of Hyves friends.

2.1.3 OpenSocial

On November 1st 2007 Google announced the release of OpenSocial - a set of common APIs for building social applications across the web - for developers of social applications and for websites that would like to add social features. OpenSocial defines a common API for social applications across multiple websites. With standard JavaScript and HTML, developers can create applications that access a social network's friends and update feeds.

Hyves is one of the first platforms supporting the OpenSocial specifications. This means OpenSocial based gadgets can be created for the Hyves platform. Hyves users can then add these gadgets to their Hyves home page or profile page. The gadgets built on the platform can access a limited amount of information from the Hyves user and his friends.

DEPARTMENT OF SOFTWARE ENGINEERING (EEMCS)
DELFT UNIVERSITY OF TECHNOLOGY

D. Requirements analysis

BuddyFuse

Yousef El-Dardiry (1332686)

10/27/2008

Table of Contents

Features

1 MoSCoW

The MoSCoW overview below divides the features of the application into must haves, should haves, could haves and would haves. The capitalized word after every feature functions as a name the features can be referenced by from other documents and code.

1.1 Must haves

1. Coupling of Hyves accounts to messenger accounts (ACCOUNTS)
2. Loading Hyves friends in Messenger (CONTACTS)
3. Notifications of activity on Hyves via alerts (ALERTS)
4. Access to Hyves user profiles from Messenger (PROFILES)

1.2 Should haves

5. Mechanism to check for updates (UPDATES)
6. Showing status of Hyves friends (STATUS)
7. Enabling chat interoperability between Hyves friends and the Messenger client (CHAT)
8. Show Hyves WhoWhatWhere (WWW)

1.3 Could haves

9. Update Hyves WhoWhatWhere when changing PSM (WWWPSM)
10. Show Hyves display pictures (AVATAR)
11. Posting scraps to Hyves users (SCRAPS)
12. Posting “tikken” to Hyves users (TIKKEN)
13. Access to a mini-Hyves website (MINI)

1.4 Would haves

14. Match Hyves contacts with Windows Live buddies and show them as a single user (MATCH)
15. “Smart messenger groups” based on profile information in Hyves (GROUPS)
16. Show latest blog posts and photos of Hyves users (ACTIVITY)
17. Add support for all XMPP networks (XMPP)
18. Integration with other social networks

2 User stories

2.1 Coupling of Hyves accounts to messenger accounts

The user should be able to associate and authenticate his Hyves account with a specific Windows Live Messenger account. He also must be able to sign in and sign out with the associated accounts.

2.2 Loading Hyves friends in Messenger

Hyves friends of the signed in Hyves user should be displayed in messenger.

2.3 Notifications of activity on Hyves via alerts

Alerts must popup when the signed in Hyves user receives a new scrap or private message on Hyves.

2.4 Access to Hyves user profiles from Messenger

A basic view of the Hyves user profiles of Hyves friends should be accessible from within the messenger client.

2.5 Mechanism to check for updates

The application should periodically check for updates at a web server. Updates should be able to be mandatory or optional, and the application should behave according to this.

2.6 Showing status of Hyves friends

When a Hyves friend changes his status on Hyves Chat, his status should be shown in the Messenger client. Vice versa, the status of the Messenger user should be reflected in Hyves Chat.

2.7 Enabling chat interoperability between Hyves friends and the Messenger client

The user should be able to open a conversation with an online Hyves friend and send and receive messages in this conversation. The user's messenger status should be reflected on Hyves, so Hyves users can also start a chat with the Hyves user using the messenger client.

2.8 Show Hyves WhoWhatWhere

Show the WhoWhatWhere message of Hyves in the Messenger client, similar to how the Personal Status Message is shown of regular users. Update this message when an Hyves user posted a new WWW.

2.9 Update Hyves WhoWhatWhere when changing PSM

When the messenger user updates his or her Personal Status Message, he should be able to select whether he wants to push it to Hyves. If so, update the user's Hyves WWW simultaneously with the PSM.

2.10 Show Hyves display picture

Load the display picture of every online user from Hyves, and show this picture in the user interface.

2.11 Posting scraps to Hyves users

The user should be able to post Hyves scraps to his friends from the messenger interface.

2.12 Posting "tikken" to Hyves users

The user should be able to send "tikken" to Hyves friends.

2.13 Access to a mini-Hyves website

The user should be able to access a small version of the Hyves website with the most important functionality inside the messenger client.

2.14 Match Hyves contacts with Windows Live buddies and show them as a single user

Instead of adding all Hyves contacts to the contact list, all Hyves buddies are matched with Windows Live contacts. If the end user already added the Hyves friend as Windows Live contact, do not add a new user, but copy Hyves functionality to the existing user.

2.15 "Smart messenger groups" based on profile information in Hyves

Let the user organize their contacts using "smart groups". These are groups based on Hyves profile information. For example, the user would be able to group contacts by high school.

2.16 Show latest blog posts and photos of Hyves users

Show Hyves blog items and latest photos of Hyves contacts similar to how contact cards show this information of regular contacts.

2.17 Add support for all XMPP networks

The user can select other IM networks based on XMPP to sign in with simultaneously. Contacts from these networks will be loaded as well, and available for instant messaging.

2.18 Integration with other social networks

Add similar functionality for other social networks such as Facebook.

Non-functional requirements

1 User documentation & usability

One of the goals of the application is *seamless integration*, which means the end-user should only notice very basic changes in the user experience. This means no end-user documentation is necessary for the application. However, a web page with frequently asked questions will be useful, to explain the product to potential users.

2 Legal aspects

Since the application is meant to be distributed via the internet, it is important it does not violate any laws / licenses. All third party source code used in the application must be checked for an accompanying license, and if it does not serve commercial purposes one must be purchased, or we have to switch to another library.

Before final release, it will also be important to write a user license agreement to accompany the installer. Probably, this process must be supervised by a law firm.

3 Compatibility

The application should be compatible with all versions of Windows Live Messenger 8 and up. It must be tested to work with 32-bit versions of the XP and later Windows operating systems. 64-bit is to be supported in a future release.

The application should also be compatible with other extensions for Windows Live Messenger, mainly Messenger Plus! Live. It should also be checked for compatibility with add-ons such as StuffPlug and Messenger Discovery.

4 Programming language

As mentioned in the PROJECT PLAN, the preferred programming language for both involved developers is C#. This language is widely adopted, and relatively easy to learn for someone with experience in object-oriented programming. Therefore, it should be relatively easy to find additional team members future if necessary.

Low level parts of the application will be written using C++ and Managed C++, if necessary in combination with inline assembly. C# remains the preferred language however, and other languages should only be used to expose functionality which cannot be implemented with C#.

5 Language

The first version of the product must support English and Dutch as user interface languages. However, during development we must keep in might future releases should be able to contain more languages.

6 User interface

The application will integrate most of its features into Windows Live Messenger in a seamless way as described above. This means for most features the user interface provided by Windows Live Messenger will be reused. If additional dialogs are found to be necessary, WinForms technology can be used. Dialogs can be designed at the beginning of development iterations.

DEPARTMENT OF SOFTWARE ENGINEERING (EEMCS)
DELFT UNIVERSITY OF TECHNOLOGY

E. Design and implementation

BuddyFuse

Yousef El-Dardiry (1332686)

1/26/2009

Table of Contents

Introduction

In this document the overall application design and implementation specifics are discussed. The first part starts by analyzing the user stories and the building blocks required for every feature to be built upon. Based on this analysis, an architectural design will be explained and the different packages are introduced. Then the overall design of the main application classes will be discussed, as well as some specifics on classes related to functionality the user stories will be built upon.

The second part discusses the implementation and design of every specific user story. Story by story, it discusses the overall idea behind that particular stories implementation as well as a short description of the related classes.

For a proper understanding of the technical parts in this document, it is important to have read the RESEARCH STUDY. A general knowledge of the Windows Live Messenger service architecture and MSNP protocol is also recommended (http://msnpiki.msnfanatic.com, http://hypothetic.org/docs/msn/).

Initial design phase

Before we start discussing the actual development iterations where every part of the software will be designed, developed and tested on a per feature basis, we take a look at the initial design. This includes looking at the overall architecture, and decomposing the application into different packages.

1 Feature dependencies

By examining the list of features in the REQUIREMENTS ANALYSIS, four common libraries can be determined where the development of these features will depend on. First of all, many of the features will depend on integrating with the Messenger user interfaces. To develop these features, a library will be required to integrate with the DirectUI UI framework Messenger has been built upon. Possibly, we also need to integrate into Win32 features used in the user interface, such as the creation of menu items.

Another key pillar of the program will be network traffic interception. Features such as adding contacts to the contact list, or enabling chat interoperability cannot be achieved using any of the exposed APIs. We can achieve this however by injecting our own data into the MSNP protocol Messenger uses to communicate with the chat servers. HTTP requests to web services can also be intercepted to inject our own data.

Both the UI integration and network traffic interception modules should make use of function detouring to achieve their functionality. This means a library should be written to expose function interception to the other modules. This functionality should be exposed to both C++ and .NET languages, so the libraries built on top of it do not have any language restriction.

The next module features will be built upon is the internal Messenger API. Although this API officially has been deprecated, we can still get a hold of it by intercepting the Component Object Model (COM) CoRegisterClassObject function, as described in the RESEARCH STUDY. In this function we can get a hold of the IMessenger interface, exposing objects, methods and events which might become useful during the development of a different number of features.

The last pillar is of course the Hyves API. A module to connect with both the HTTP based Hyves API and the Hyves Chat platform will be necessary to integrate Messenger with Hyves. Most features will depend upon at least one of these four domains. In Table 1 on the next page, the expected dependencies of every feature are shown.

Depends upon / Feature	UI Integration	Network traffic interception	Hyves API	Messenger API
1. Coupling of Hyves accounts to messenger accounts				
2. Loading Hyves friends in Messenger				
3. Notifications of activity on Hyves via alerts				
4. Access to Hyves user profiles from Messenger				
5. Mechanism to check for updates				
6. Showing status of Hyves friends				
7. Enabling chat interoperability between Hyves friends and the Messenger client				
8. Show Hyves WhoWhatWhere				
9. Update Hyves WhoWhatWhere when changing PSM				
10. Show Hyves display pictures				
11. Posting scraps to Hyves users				
12. Posting “tikken” to Hyves users				
13. Access to a mini-Hyves website				
14. Match Hyves contacts with Windows Live buddies and show them as a single user				
15. “Smart messenger groups” based on profile information in Hyves				
16. Show latest blog posts and photos of Hyves users				
17. Add support for all XMPP networks				
18. Integration with other social networks				

Table 1, dependencies of features upon four main domains, a green cell indicates a dependency, and a yellow cell a possible dependency

One of the first things one might notice when looking at Table 1, is not many features depend upon the original Messenger API. Although not a lot of features might strictly depend on it up-front, it is likely features of the API will be used throughout the code for scenarios such as looking up contacts or conversation windows. This should become clearer when we advance to the first development iterations.

2 Architectural design

In this part an architectural overview of the design will be discussed. Next, the system will be divided into different packages, and these will be briefly introduced.

2.1 Overview

Figure 2-1, shows us an architectural overview of the system. First, there is the framework level. The main components will be built on top of the .NET Framework. However, since the original Messenger API is built on top of the Component Object Model object based-framework, this is also listed as one of the platforms. The Win32 APIs will be used for low-level integration with the Messenger client.

On top of the underlying frameworks, a hooking library for function interception must be built. This library will expose functionality to intercept function calls made by the Messenger application to other modules.

On the next level are the modules which will be directly used to build features upon. These are the four domains introduced in Feature dependencies above. All modules except the Hyves API module will require the functionality exposed by the hooking library to implement their features.

The UI integration module can be separated into two subsets, a part to integrate with the Messenger DirectUI library and a part to integrate with Win32 UI functions. The network traffic interception module must intercept both Winsock and WinInet communication, described in the RESEARCH STUDY. For the integration with Hyves, both the chat platform and HTTP-based API will be used.

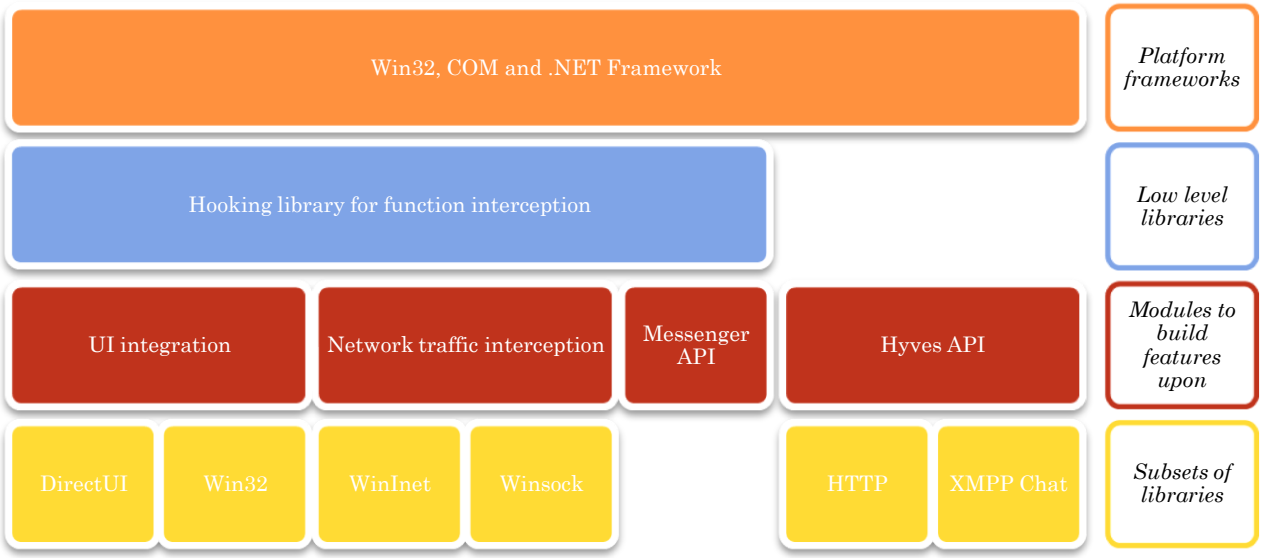

Figure 2-1, an architectural overview of the application

2.2 Packages

Now the different modules required to implement features upon have been identified, packages can be defined to decide where to locate every module. Every package will correspond to a project in the Visual Studio Solution, and will be built separately. This means eventually every package will be located in a different DLL library or executable.

One constraint in defining packages is the language used to write different modules. C++ code cannot be combined with C# code in the same package. C++ code can however be mixed with Managed C++ (MC++) to enable integration with the .NET Framework. Combining C++ and MC++ will be referred to as Mixed Code.

In this section four different packages will be introduced, namely the Loader, Core, UIWrapper and Hooks packages. As a reference, their dependencies upon each other are shown in Figure 2-2 below.

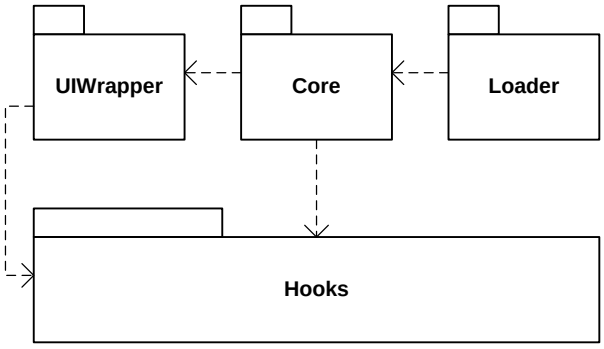

Figure 2-2, package dependencies

2.2.1 Hooks

As can be seen in Figure 2-1, the hooking library for function interception will be the basis for most of the functionality in the application. This library must expose functions to enable function interception, both to C++ and .NET based languages. Since feature interception is an advanced task requiring low level language functionality, this library will be written in C++ (if necessary Inline Assembly will be used as well).

2.2.2 Loader

One area not discussed as of yet, is how our extension application will be loaded by Windows Live Messenger. To integrate our application into Messenger, our libraries should be loaded into the Messenger process. This can be achieved in a couple of different ways, as described in the RESEARCH STUDY. The most suitable method to use would be to use a proxy DLL. In this way we do not need to monitor the system to see when the Windows Live Messenger process (msnmsgr.exe) is started, since our library will be loaded automatically when it does.

To achieve this, the proxy DLL must be written. Because this DLL will be responsible for loading the rest of our application, we will call it the Loader DLL, or package. It must be written in C++, because .NET code cannot be loaded during the startup of a process. For more information on this refer to (Microsoft Corporation, 2007).

2.2.3 Core

The actual user stories / features will be implemented in one module. Since this module will be the one where most of the application logic resides, this module will be named Core. As described in the REQUIREMENTS ANALYSIS, C# is the preferred language for use in this project. The Core module will therefore be written in this language.

Functionality that cannot be written in C# will reside in one of the other packages. With Managed C++, wrappers can be written to expose native functions to any .NET language. This way native functions can be used in the Core project. Another method to accomplish this is defining native functions directly in C# using P/Invoke (Platform Invocation Services).

2.2.4 UIWrapper

To implement the user interface integration module, introduced above, our application must integrate with the DirectUI user interface framework used by Messenger. In the RESEARCH STUDY, we found out this library resides in one or more DLLs in the Messenger installation directory. Since the DirectUI framework is a framework written in native code, we must find a way to interact with it from our managed code Core module. This can be accomplished by creating managed wrapper classes in Managed C++ around the required native classes and methods exported by the DirectUI library.

These wrapper classes will have to reside in a different package than the core application functionality since it must be written in Managed C++ and C++ (mixed code). We will name this wrapper package UIWrapper. It will only be necessary to wrap functionality required by one of the other packages, and will therefore grow incrementally as more user stories are going to be implemented.

3 Core

This part introduces the design overview and class overview of the main package of the application, *Core*.

3.1 Design overview

The Core module's main task is to bind the Hyves and Windows Live Messenger services. To implement this, classes must be implemented to interact with both of these services, and to present information about the application's state to the user. Based on this idea the components in Core can be divided into three categories. The first category consists of components exposing functionality of other programs / services (Hyves, Windows Live Messenger), these components will be referred to as APIs because they expose programming interfaces of existing programs. The other categories consist of components interacting with the user interface, and the main components to tie the others together.

3.1.1 Controllers

Although the APIs could talk directly to each other, we have chosen to use a design based on the mediator-pattern to loosen coupling (Wikipedia contributors, 2008). In this design, all functionality depending on one or more of the APIs will be represented in its own *Controller*, which talks to the APIs. By enforcing loose coupling between components it should be easier to add new features to the program, for example enabling support for other social networks.

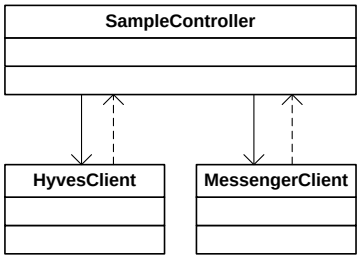

Figure 3-1: communication between APIs via a controller. Solid lines indicate direct communication, and the dashed lines indirect communication (via events).

Every controller has a single responsibility to control a particular function or coupling of one or more APIs. For example, the *SampleController* in Figure 3-1 could be responsible for logging out the HyvesClient user whenever the MessengerClient user logs out.

By following this single responsibility principle, we prevent the need of having a difficult to maintain bloated manager-class binding the different components of the application together. Every controller is expected to be a relatively lightweight class, which makes them easy to test and maintain.

3.1.2 User interface

As described in the REQUIREMENTS ANALYSIS, the application tries to extend the Windows Live Messenger application in a seamless way, which means most of the features are expected to become visible to the user via the existing interface. It is expected only small dialogs and buttons will be added to the application, which means the UI layer of the application will be relatively small.

Based on this expectation it seems reasonable for the user interface components to interact with the API layer in much the same way as the controllers do. User interface patterns such as the MVC (model-view-controller) or MVP (model-view-presenter)

pattern seem inapplicable because in most cases the actual View part of these patterns will be provided by Windows Live Messenger itself. By giving the user interface layer a little more power than conventional in most patterns we expect to save a lot of time that would be spend on enforcing one of these patterns. The saved time is paid for by a slight decrease in the loose coupling of the user interface layer. However, since this layer is expected to be small, this price seems reasonable.

3.2 Class overview

With the overall design in mind, we can now discuss a basic class diagram showing the relations among the most important classes to be implemented (Figure 3-2). Note that this is a conceptual diagram of what the classes and their relationships will look like. It is expected changes will be made during the development iterations, since we have chosen not to design everything in advance, but to design incrementally during the development iterations instead.

3.2.1 Mashenger and IntegrationManager classes

"Mashenger" will function as a temporary name for the project, and also be the namespace the classes will reside in. The Mashenger class will be the entry point for the extension, and will be initialized by the Loader. The entry point will load the IntegrationManager class which manages the initialization of the actual extension.

3.2.2 Window classes

Similar to other window-based applications, every window will be represented in its own class. The difference here is the actual design of the window has been implemented by Messenger already, so these Window classes will only contain the logic added by the extension. The Window classes used (in the diagram MainWindow, LoginWindow and ConversationWindow) all inherit from the abstract Window class which keeps a relationship with the original DirectUI window element.

3.2.3 HyvesClient and Contact classes

The HyvesClient class manages communication with the Hyves API and Hyves Chat. For this functionality third party libraries will be used, namely Bee.NET for the Hyves API, and jabber-net for communication with the chat server.

The HyvesClient class keeps track of the friends of the signed-in Hyves user represented in HyvesContact classes. The HyvesContact classes derive from a Contact class so other chat networks can be supported in the future. For the same reason HyvesClient implements an IChatService.

3.2.4 MessengerClient

The MessengerClient class is the API module exposing functionality required to integrate with Windows Live Messenger functionality unrelated to its user interface. It utilizes the COM-based Messenger API (discussed in the RESEARCH STUDY) as well as the network interception classes discussed below.

3.2.5 Network interception classes

Interception of HTTP(s) requests is taken care of by the WebProxy class, it raises events to the Mashenger class whenever it intercepts an outgoing request.

Interception of WinSock traffic is taken care of by the Socket and SocketManager classes. The NotificationManager and Conversation classes use a Socket object to intercept network traffic to the main Messenger server (Notification Server) and conversation servers (Switchboard Servers).

3.2.6 AccountManager and UserAccount classes

The user account related classes contain information about accounts used by the end-user. The AccountManager class manages the collection of activated chat clients associated with different user accounts.

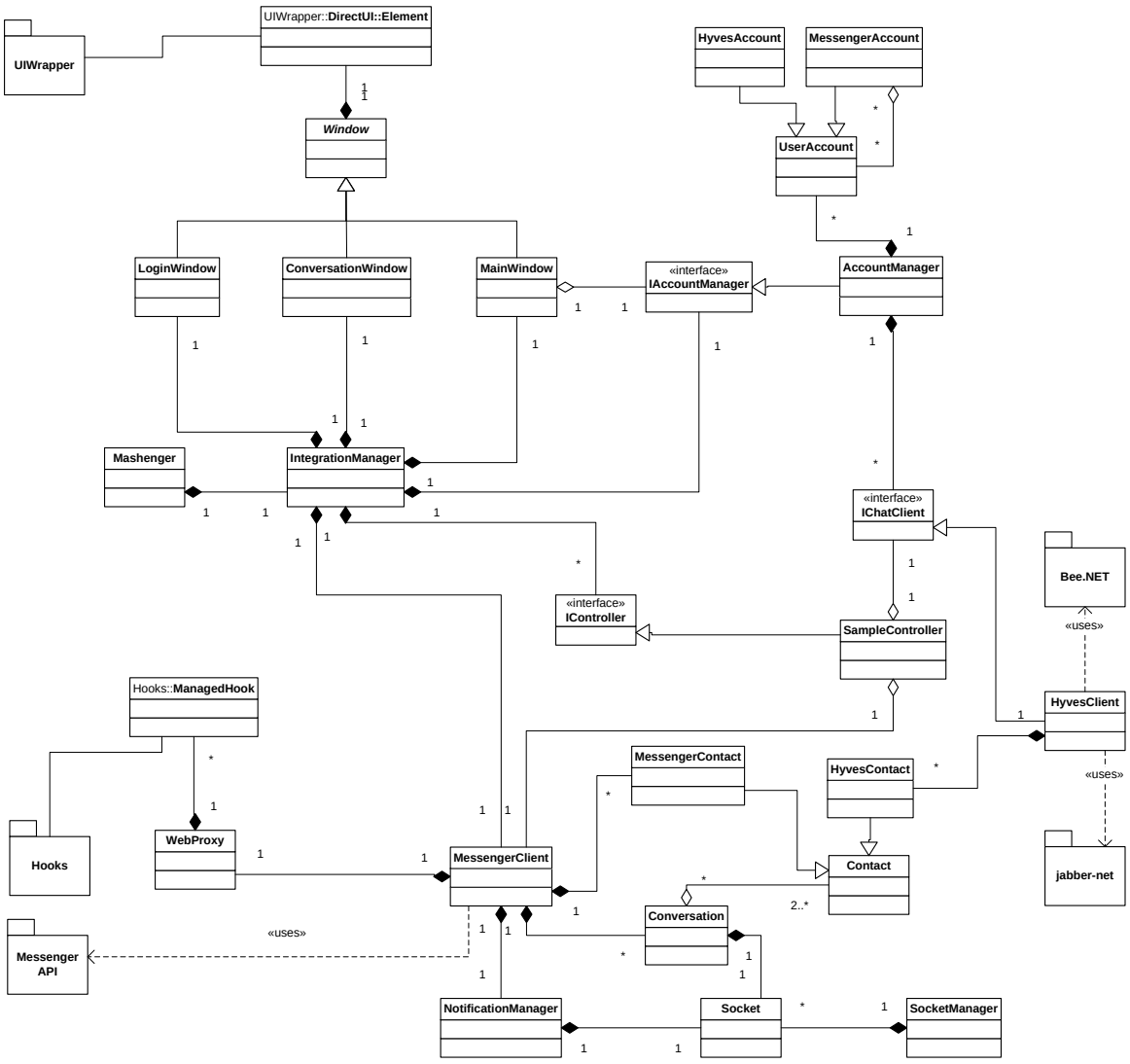

Figure 3-2, class overview of the Core package

3.3 Protocol interception design

Perhaps one of the most complex parts of the Core module is the design of the interception of MSNP (communication protocol used by Messenger) traffic. Messenger communicates MSNP messages with the Notification (NS) and Switchboard (SB) chat-servers either directly over TCP sockets, or indirectly via an HTTP gateway server. In this part we will discuss how both communication channels are intercepted.

3.3.1 Direct Winsock connections

Almost in every configuration, the Messenger client will communicate with the Messenger NS and SB servers directly over a TCP connection. This traffic can be intercepted as described in RESEARCH STUDY (intercepting network traffic, Winsock). In this part the class design of the Winsock interception will be discussed.

Note that the current design only supports asynchronous sockets configured with the WSAAsyncSelect function.

NetworkInterception.Socket

The Socket class represents a single Winsock socket. It exposes events indicating data is being read from or written to the socket. When another component handles these events it can choose whether or not to forward them to the real socket object, and modify data sent / received if necessary. The Socket class also exposes an event indicating the socket is being closed.

Arguably the most important function of the Socket class is to be able to trigger the host application to read from the socket, and then inject custom data. This functionality is exposed via the *ReceiveData* method.

NetworkInterception.Socket.SocketMonitor

The SocketMonitor class registers hooks to intercept function calls to relevant Winsock functions (WSAAsyncSelect, recv, send, close and connect). In most cases it then notifies the targeted Socket object of this call (it keeps track of a collection of Socket objects). SocketMonitor is implemented as a nested class inside Socket so it can forward calls to intercepted functions to private methods of the Socket class (similar to a C++ friend class).

Networks.Messenger.MessengerSocketMonitor

A Messenger-specific SocketMonitor, which exposes events for the creation of a Socket connected to either a switchboard or notification server.

3.3.2 Gateway-relayed connections

When the Messenger client cannot directly connect to the chat servers (for example, due to firewall restrictions), it switches to what we will refer to as *gateway mode*. In this case it connects to a HTTP web server which functions as a proxy to the actual chat servers. The Messenger client polls to this web server at a regular interval to see whether any new MSNP data is available (Mintz). This technique is similar to what numerous AJAX enabled web applications use to simulate socket connection using HTTP requests. As the matter of fact, the specific HTTP gateway servers are also being used by web-based Messenger API's such as the Windows Live Messenger Client Library.

To make sure our extension supports all configurations of Messenger, it must also support gateway mode. To develop the support for this configuration, one can block outgoing TCP connections to a server with port 1863 (used by NS and SB servers) in his firewall to force the Messenger client to use the HTTP gateway.

Since the HTTP gateway connections basically represent a socket connection, we can base the class design of gateway mode interception upon the design used for direct Winsock connections. The difference in technical implementation is that we need to intercept WinInet traffic as opposed to Winsock traffic. This can be accomplished as described in the RESEARCH STUDY (intercepting network traffic, WinInet). Below are the classes relevant to gateway mode socket interception.

NetworkInterception.ISocket

Now we have identified two types of socket connections, their common operations can be defined in the ISocket interface. An ISocket represents a data connection exposing events when data is sent to and read from the underlying connection. Also, data can be injected into the connection.

NetworkInterception.Socket also implements ISocket.

Networks.Messenger.GatewaySocket

GatewaySocket is the ISocket implementation for HTTP gateway connections. It requires a WebProxy object as source for the HTTP traffic it operates upon.

Networks.Messenger.GatewaySocketMonitor

The GatewaySocketMonitor monitors a WebProxy to see whether new gateway connections to a NS or SB server are created. When they are, it creates a new GatewaySocket and fires an event to notify listeners about this instantiation.

3.3.3 Using ISockets

Now the classes to intercept MSNP connections have been defined, let us see how they can be used to expose functionality to other classes.

Networks.Messenger.NSConnection / Networks.Messenger.SBConnection

These classes operate upon an ISocket object connected to a notification server or a switchboard server respectively. Both classes expose functionality to other classes which requires them to insert or monitor MSNP protocol data transmitted through the socket.

User stories implementation

1 Coupling of Hyves accounts to messenger accounts

The sign in experience can be seen as the "entry point" of the application extension in the eyes of the end-user. The end-user must be able to authenticate his Hyves account so the Hyves API layer can make requests to the Hyves service. This user story can be divided into two main parts:

- Managing associated Hyves accounts
  - Adding and authenticating a Hyves account
  - Remove an associated Hyves account
- Sign in / sign out from an associated Hyves account
  - From the messenger main window
  - From the messenger login window

Hyves accounts can be associated with a particular Windows Live ID. A user can only manage/add his associated accounts after he has been signed in. This prevents other users from changing accounts associated with that particular Live ID. When associating a new Hyves account, the user must be directed to an authorization page on the Hyves website.

Once an account has been associated with a messenger account (Live ID), the user can choose to sign in with this account. This can be done from the main window, as well as from the login window.

1.1 User interface

To present the above functionality to the user, a couple of modifications must be made to the messenger interface. A button is added to the messenger main window from where a dropdown menu pops up. From this dropdown menu, the user can select to manage his accounts, and sign in to / out from already associated accounts. When he selects to manage his accounts, a dialog will pop up similar to the one shown in Figure 1-1.

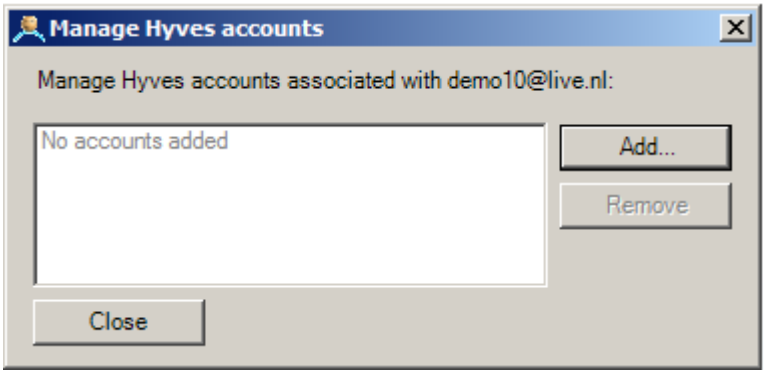

Figure 1-1: the manage accounts dialog

Pressing the Add-button will pop up a form hosting the Hyves website API authorization page where the user can select to allow our application to sign in on his behalf. After authorizing a Hyves account, the option will be added to sign in / out from this account from the Hyves-button in the main window.

Also, when the same Live ID account is selected in the messenger login window, an option will be shown to sign in with an associated Hyves account simultaneously (Figure 1-2).

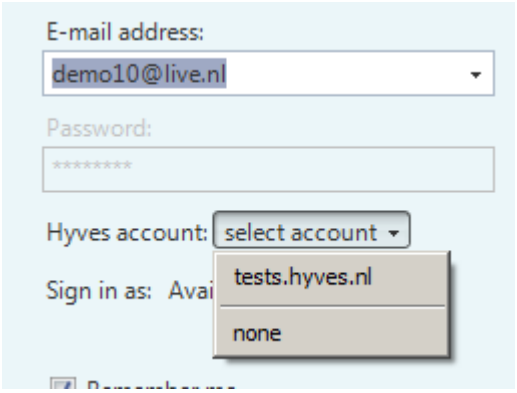

Figure 1-2: simultaneous sign in

1.2 Class implementation

Below the tasks of different classes are described to implement the ACCOUNTS user story.

Networks.Messenger.MessengerAccount / Networks.Messenger.HyvesAccount / Networks.AccountDictionary

Classes containing no logic, but only information about the associated accounts. These are serializable so the information can be persisted in the user settings. ManageAccountsForm provides a user interface to operate on the data provided by these classes.

Networks.Hyves.HyvesAccountDialog

WinForms implementation of the dialog showing the Hyves API authorization page. Upon success, this provides a new HyvesAccount object the user authenticated with.

Networks.Hyves.HyvesClient

HyvesClient is the main class responsible for communication with the Hyves API and Hyves XMPP server. It exposes a sign in and sign out method to be called when the user clicks the corresponding button, or when an account is selected in the main window for simultaneous sign in.

Networks.AccountManager

Manages the mapping between UserAccounts and IChatClients (HyvesClient). Exposes methods to activate and deactivate a user account, and events objects can subscribe to, to be notified of chat client activation / deactivation.

Controllers.LoginController

Binds HyvesClient and AccountManager so the HyvesClient signs in whenever the corresponding user account is activated, and signs out whenever it is deactivated.

UI.LoginWindow

Adds the associated account information similar Figure 1-2.

UI.MainWindow

Adds the button and corresponding dropdown menu to the main window, as discussed above.

UI.ManageAccountsForm

The form to manage associated accounts, similar to the one shown in Figure 1-1.

2 Loading Hyves friends in messenger

Now Hyves accounts can be associated with a messenger account and the user can choose to sign in with one of these associated accounts, we can continue with the integration of this account with Messenger. The first step is to find a way to show the Hyves contacts of the signed-in Hyves account in Messenger.

The ideal way to accomplish this would be to show the Hyves contacts inside the Messenger buddy list, since this would be the most seamless way for the end user.

2.1 Injecting contacts

Unfortunately, neither the Messenger API or DirectUI modules expose an easy way to inject contacts to this buddy list (without adding them to the Windows Live address book as well). However, it is possible to trick Messenger into thinking the currently signed in user has added Hyves contacts to its address book.

With a tool such as HTTPAnalyzer, we can see Messenger uses HTTP SOAP requests to synchronize its address book with the central Windows Live ABCH (Address Book Clearing House) servers. For example, when a user adds a contact to his address book on Live Hotmail, Messenger will receive a notification that the address book has a new update. It will then contact the ABCH to receive the latest synchronization information.

Using the techniques described in the RESEARCH STUDY on WinInet hooking, it is possible to intercept the ABCH requests and inject additional contact information.

2.1.1 wlcomm.exe

Since the version 9 beta of the Windows Live Messenger (released in 2008), the ABCH SOAP requests are not carried out by the Messenger process itself anymore. Instead, a separate process, the Windows Live Communication Platform (wlcomm.exe) takes care of synchronizing the address book with the ABCH. This means that we now must also inject into this process to intercept its HTTP requests.

2.2 Class implementation

Below the tasks of different classes are described to implement the CONTACTS user story.

Networks.Contact / Networks.Hyves.HyvesContact

The abstract Contact class is a unified way to represent a contact across the application. It exposes the information required by MessengerClient to interact with contacts. HyvesContact is the Hyves-specific implementation of the Contact class.

Networks.IChatClient

IChatClient exposes an event to indicate contacts of the signed in user have been loaded, and a property to retrieve the loaded contacts.

Controllers.ContactController

The ContactController controller is responsible for loading contacts of a chat client to the messenger client.

Networks.Messenger.ContactManager

Loads and removes additional contacts into Messenger. Loading of contacts is achieved by listening for proxy requests related to contact synchronization. Contacts are removed directly via the Messenger API. The Messenger API is also used to monitor whether Messenger has loaded / unloaded a contact from the buddy list.

NetworkInterception.WebMonitor

WebMonitor monitors WinInet web requests made by the application. It redirects requests to the ABCH to a local address.

NetworkInterception.WebProxy

WebProxy is a lightweight HTTP server where HTTP requests can be redirected to (by WebMonitor). It fires an event for every incoming requests so the received request or response to return can be modified.

Mashenger

The "startup" Mashenger class must now also instantiate a WebMonitor when it is loaded into the wlcomm.exe process.

3 Showing status of Hyves friends

Now Hyves contacts are loaded inside the Messenger buddy list, we want to make sure their statuses reflect their status on the Hyves XMPP servers. To accomplish this, we need to be able to trick the Messenger client into thinking one of the contacts changed his or her status. This can be achieved by letting the active NSConnection insert a contact status changed message for the affected contact.

The other way around, we must make sure that the IChatClient signed in user always reflects the status of the signed in Messenger user.

3.1 Class dependencies

Networks.IChatClient / Networks.Hyves.HyvesClient

Exposes an event to notifiy handlers one of the contacts of the signed in user changed his or her status. HyvesClient now listens to a XMPP event for presence information to implement this feature.

Controllers.StatusController

The StatusController monitors an IChatClient for status change events and forces the MessengerClient to reflect these status changes. Vice versa, the IChatClient is told to change its status when the Messenger user changed his or her status.

Networks.Messenger.MessengerClient

Because of the asynchronous nature of contacts injection (CONTACTS), MessengerClient must make sure that once Messenger successfully added a contact, its initial status will be reflected properly.

MessengerClient now also triggers an event when the Messenger user changed his or her status.

Networks.Messenger.NSConnection

The NSConnection exposes a function to insert a contact status change MSNP command (NLN) into the Notification server socket stream.

4 Enabling chat interoperability between Hyves friends and the Messenger client

Since the status of Hyves contacts is now being reflected in the Messenger buddy list, the user should be able to commence a conversation with his online Hyves friends and vice versa. This can be accomplished by intercepting MSNP messages responsible for setting up and maintaining a conversation.

In MSNP, a connection to the switchboard (SB) server represents a conversation connection. Let us say a user (Alice) wants to start a conversation with one of her friends, Bob. This scenario is illustrated in Figure 4-1; Alice first requests a new switchboard connection from the notification server (1, 2). When she has authenticated with the switchboard given to her, she can invite her friend Bob (3). Bob will be notified of this invitation via the existing connection it has to the notification server (4). He can then respond to this notification by connecting to the same switchboard session Alice is in (5). The switchboard then notifies Alice Bob has joined the conversation, and from now on the two can send messages to each other.

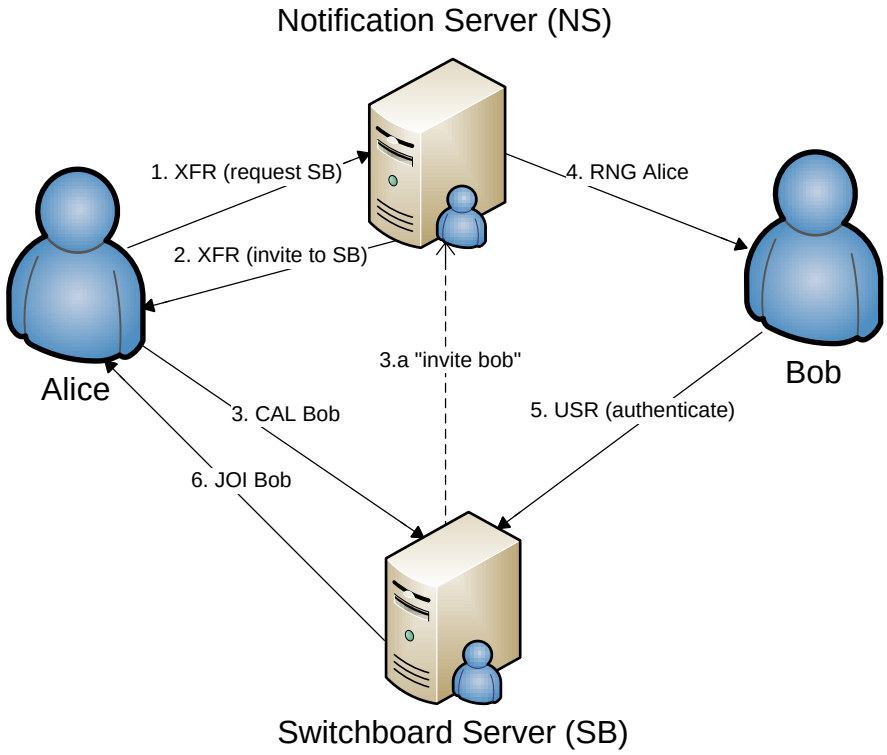

Figure 4-1: a schematic overview of how conversations are established in MSNP. In this scenario, Alice starts a conversation with Bob.

By examining the figure we can determine two scenarios our application must be able to handle. In the first scenario, the end-user starts a conversation with a Hyves contact. Our application can see this when step 3 in the figure is executed with a Hyves contact as parameter. From this point, the application must "hijack" the switchboard server connection and act as if Bob has entered the conversation. It must then monitor for conversation messages and forward them to the right endpoint.

In the other scenario a Hyves friend starts a conversation with Bob. We want to force the Messenger client to show a conversation window with this message, so we must

inject data simulating the MSNP switchboard invite (step 4). Of course, we must also accept the connection Bob will attempt to make to the switchboard and forward any messages sent to and from it.

4.1 Class dependencies

Functionality added to classes specifically for this user story is described below. Most functionality has already been described above in the protocol interception design.

Networks.IChatClient / Networks.Hyves.HyvesClient

Exposes an event to notify handlers of incoming conversational messages. HyvesClient now listens to a XMPP event for message information to implement this feature. It also exposes a method to send a message to a particular contact.

Networks.Messenger.MessengerClient

Likewise, the MessengerClient class exposes methods to receive a message from a particular contact, and an event when the user sends a message to an injected contact.

Controllers.ConversationController

The ConversationController monitors an IChatClient for incoming messages events forwards these to the MessengerClient and vice versa.

Networks.Messenger.ConversationManager

The MessengerClient class holds a reference to a ConversationManager. This class is responsible for monitoring and managing SBConnections.

Networks.Messenger.SBConnection

The SBConnection object encapsulates a socket connected to a switchboard server. It monitors MSNP messages related to conversations and exposes related functionality to other objects.

Networks.Messenger.NSConnection

The NSConnection exposes a function to trick Messenger into connecting to a switchboard server, as described above.