Framework for on-demand delivery of data and automatic patching of code and content.

June 2007

Aalborg University

AALBORG UNIVERSITET
DEPARTMENT OF COMPUTER SCIENCE, FREDERIK BAJERS VEJ 7E, DK 9220 AALBORG ØST

Title: Framework for on-demand delivery of content and automatic patching of code and content.

Project period:
Speciale (Dat6)
February 1th 2007 - June 11th 2007

Project group:
E4-117

Group members:

Kammersgaard, Marc

Melsvik, Jasper

Nielsen, Rasmus

Supervisor:
Vestdam, Thomas

Copies: 7

Page count: 131

Abstrakt:

The rapid development of mobile phones has increased the use of mobile phones as gaming devices. In this project we will focus on easier development of mobile games, with regards to content and patching of code. To help mobile game developers we have created a framework called FoDa which handles on-demand delivery of data and automatic patching of code and contents. Handling this developers do not have to think about were game content comes from, how it should be saved and processed. The developer only has to request the data needed and FoDa will provide the data. As mobile development have a short time-to-market requirement, changes in the design of a development project require the framework to be highly flexible. To make FoDa as flexible as possible we have produced the framework by using a pure plugin architecture. This gives the developers the possibility to tailor FoDa to the specific needs of a project. This report will describe the process from an idea of the framework to the implementation and evaluation of the framework.

Table of Contents

Chapter 1

Introduction

Recent years mobile devices and in particular mobile phones have become wide spread. Statistics from the National IT and Telecom Agency in Denmark shows that mobile phone subscriptions in Denmark in the year 2005 exceed the number of people living in the country [1]. Not only Danes are fond of mobile devices, according to the article "500 million Chinese have a mobile before summer" [2] featured in Computerworld. The number of phone subscriptions increase with 13.5 million from February to March 2007. The majority of these phones are multimedia enabled, able to use images, sounds and videos, which make ideal gaming devices.

Devices are rapidly becoming smaller, faster and more powerful in terms of their capabilities. This has the effect that devices vary greatly making production of software, which supports a large range of devices, a hazel.

Demand and supply of mobile games are rapidly increasing as more people use the devices for causal gaming.

According to the Mobile Game Conference 2006, did the revenue in the US from mobile downloads increase from $40 million per month to $100 million per month from November 2005 to May 2006 [3]. In the US games downloaded represent about 6% of the mobile data revenues, where it in Japan is 15-20%. Mobile application companies are starting to see business in developing games for mobile devices, but game development on mobile devices often suffer from short time to market requirements which implicates short development cycles. This causes the side effect of cutting innovation corners by copying existing concepts from games which has proven themselves in the early arcade games or porting of console and PC games. This is more thoroughly described in the report "Problems related to massive multiplayer online game development for mobile phones" [**?**]. Even the most attractive game genre within the PC market is starting to rub of on the mobile devices game concepts, as the all famous Massive Multiplayer Online games starts to surface. Firstly with the mobile Roleplay game Tibia [4].

Solutions to remedy the diversity of hardware platforms exist in form of virtual machine layers among others: Java Micro Edition and Microsoft .Net Compact

Framework. These frameworks state to ease development of application across devices by the *code once, run everywhere* philosophy. They also provide easy access to features such as display devices, input devices, network capability and other hardware feature. The problem that still remains when using these frameworks is that a large amount of work needs to be done to practically support features in the application. For example network connections still require working with sockets if one is to support a particular services not running the HTTP protocol. Nor do they provide features as cache or security, which are needed in many application which regular communicate through the Internet.

Another problem with mobile devices is the maintenance of software. Applications for mobile devices are rarely maintained, as it is quite troublesome to maintain mobile software, either the user have to receive a new version of the softwareand update the application himself or send the device to the company which sold it and have them update it, spending weeks without the device. Many online games do not have a clear end, *Game Over* or *Game Completed*, but is meant to be played forever, users will expect updates on a regular interval or they will become bored and stop playing the game. It is definitely not a practically solution to spend weeks without the device, every time an update arrive.

Most mobile application is distribute as one complete package, which do not allow updates of certain parts of the system, but demands that the complete package is reinstalled every time an update is released. This is particularly a problem with small changes and updates, such as bug fixes or modification of application content.

Content handling is also a problem the developers have to deal with in particular when doing game development. Mobile devices have limited disk space so strategies for optimizing its use are required.

A solution to make mobile applications and in particular mobile game development easier and faster is to create a system that simplifies tasks such as network connections and content handling. Automates the process of content delivery and automates the maintenance of applications by the use of an automated patching system. All which are intended to make it easier for the developers to develop and maintain application as well as benefit the users with more diverse and feature rich applications that are always up to date.

1.1 Problem

The problem we aim to solve with this report is outlined in the introduction. A system for increasing the ease and efficiency of developing and maintaining applications and in particular games for mobile devices is needed.

To increase efficiency of development, the system must be highly flexible and modular to support reuse as well as making complex features which require many lines of code as transparent to the developer as possible. An example of complex

code is the use of network traffic; this should not have to be a concern for the developer at all.

To ease development simple interfaces from the system to the developer is required. All types of data content stored locally or not, should be reachable from a simple interface which is easy to use.

The system must support ongoing development of the applications trough automatic patching of code and content. To give mobile devices with limited disk space the same opportunity to use the system as big devices, on-demand delivery of data will also be a necessary feature.

Important to the system is that it takes into consideration that it is to run on a mobile device with limited resources, so the footprint and resource use of the system should be as small as possible. The system should also be able to handle the characteristics of mobile application; that they are on the move, which could result in temporary inability to access services and resources.

In short we will be focusing on developing a system for mobile devices which handle on-demand delivery of data as well as automatic patching of code and content. We will implement the system as proof of concept as well.

1.2 Report structure

Section 1.1 on the facing page defined the problem which we aim to solve, but before it is possible to develop a system for mobile devices we have to be sure what features our system should contain. To make a feature list which the system should handle and requirements to the system, we will in the next chapter describe different mobile application which all have different demands to a system like ours.

After we have reviewed the different mobile application and come up with a feature list, we will compare this feature list with other existence systems to see if any system meet the demands we seek, this part is described in Chapter 3.

Chapter 4 describe our requirements to the system, which areas we will focus on and which areas we will rate as a low priority. With our feature list and focus areas in mind, Chapter 5 describe the architecture of the system. Knowing which our requirements and the architecture we should use it will then be possible to design our system to support the feature described in our feature list, Chapter 6 will describe how we will design the system. After describing all parts needed for the system; architecture, design and the engine we will in Chapter 7 describe how we have implemented the different parts of the system. After we have implemented the system we will in Chapter 8 evaluate the system, to see if it meets our requirements. Finally in Chapter 9 we will conclude upon our system and the work we have done, describing what we have achieved. Code and binaries of the project can be found on the website: http://www.cs.aau.dk/~r10/System/ until July 2007.

Part I

Research

Chapter 2

Use Cases

In the previous chapter, we described the problem we wish to solve, but before it is possible we have to know what the system should be able to do. To find a list of features our system should provide a user; we will in this chapter present a selection of use case scenarios of applications. The application will cover a range a range of applications we predict will become more popular on mobile devices as faster and cheaper networks take hold of the market.

Each use case describes the general idea of the system in question. Since the system we are developing primarily deals with delivering of data, each use case will in details describe the data flow of the system in question. After describing the use case and their dataflow, we will describe which features the use case could make use of from our system.

2.1 Digital Photo Album

This section describes the use case covering a digital photography album running on a mobile device.

Idea

A digital photo album is an application with the main purpose to display digital photos. The idea is that digital photos are stored in an archive and the user can then use the application to browse and view the photos. An example of a layout for a mobile photo album is shown in Figure 2.1 on the next page. The application displays one photo at a time and the two buttons *back* and *next* are used for navigating back and forth in the whole photo series.

The name of the photo is displayed beneath the photo and between the two navigation buttons.

The above application can easily be expanded to retrieve the images over the air. This results in a request protocol where the application will request the next

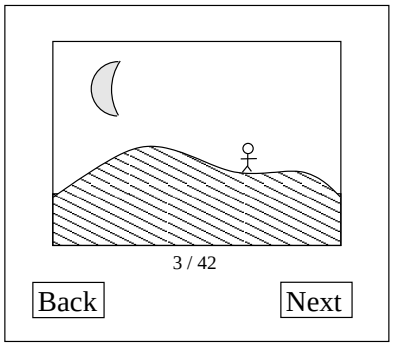


Figure 2.1: Layout of the simple image viewer.

picture from the server, if it is not already stored on the mobile device, and then store it using a cache. Concerns about security must be considered when dealing with data delivered over the air. Since users might not want others to see their personal photos, a password is required to access the photo album.

This simple application is one of the applications that our system must be able to support. To describe the data flow we first describe a common usage scenario for this application.

Usage

The user starts the application and browses from photo one trough three, using few seconds to look at each photo. Since it is not the first time the user use the application, photos one and two are already locally stored on the mobile device, but the rest are not. Due to disk space restrictions the mobile device, limits the number of locally stored photos to five.

The next section describes the data flow involved in the operation of the application.

Data Flow

Figure 2.2 shows the data flow of the usage scenario described above.

The user starts the application and the first photo is requested from the cache which hits and returns the photo for displaying. The user presses the *next* button and the application request photo two for displaying, which is also found in the cache and returned to the display. *Next* is pressed again by the user and the application requests photo three from the cache. However, photo three is not resident in the cache and the cache therefore requests the server for the photo, using the predetermined identification and password. The server validates and returns photo three to the cache which stores it and send the photo to the application for displaying.

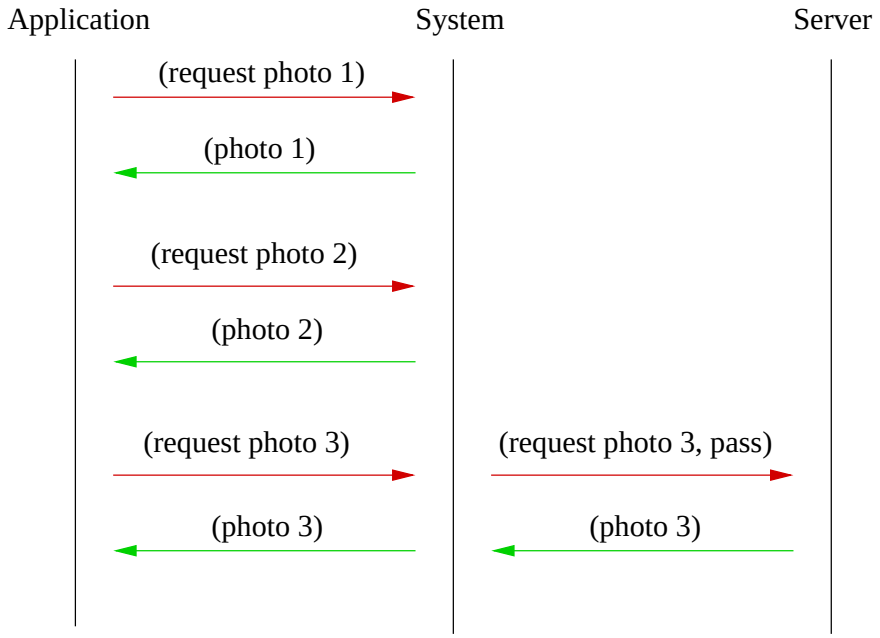


Figure 2.2: The data flow of the usage illustrated.

Requirements

This section summarizes the requirements of the digital photo album in relation to a system facilitating easy development of this type of application.

A system must be able to support network connections and the ability to deliver data fetched from the network to the application above as well as caching of images when downloaded for faster browsing. User identification is also needed.

1. Managing connection to a network
2. Delivery of image data to the application
3. Managing data cache
4. Handle user identification and security

2.2 Indoor Navigation

Today we have lots of navigation tools that help users find their way around cities. Often these systems utilize the Global Positioning System (GPS). These systems are often accurate to within a few meters. This is however not good enough for indoor navigation, like navigating in buildings with rooms and hallways, where a meter is often a large percentage of the overall area you are trying to navigate.

One of the alternative ways of indoor navigation utilize the still increasing number of Wi-Fi spots in buildings to determine location.

As oppose to GPS that make use of signal from three or more satellites to triangulate positions, indoor Wi-Fi navigation works by mapping out the Wi-Fi spots and measuring there signal strengths in different parts of the building and storing this data in a database. This mean the more measurements the more accurate the positioning will be. If users need to know their position, they have to measure the Wi-Fi signal strength and compare it to the data in the database. From this it is possible to extrapolate their position.

Idea

The idea is that an application running on the mobile device will measure Wi-Fi signal strength continuous and keep a map on the mobile device with the users location up-to-date.

The application will get map data and measurements data relevant to the user's current position from the server. The data will be streamed to the application on-demand. Zone techniques to determine what maps to deliver to the application can be used, as seen in FPS games and MMO games where the content of adjacent zones are delivered and loaded to increase response times of the application.

To further minimize the response time, the application could try to anticipate what will be updated next. An example would be to create a grid of the building map and determine which blocks of the grid the user is likely to enter and load those before the user enters.

As storage space is assumed to be limited on the mobile device only the portions of the map of the buildings relevant to the user are stored on the device together with signal strength data relevant to the maps stored.

This should make the mobile device capable of doing the extrapolations required itself. Speeding up the process of determining the locations of the user.

Figure 2.3 show an example layout of this type of application.

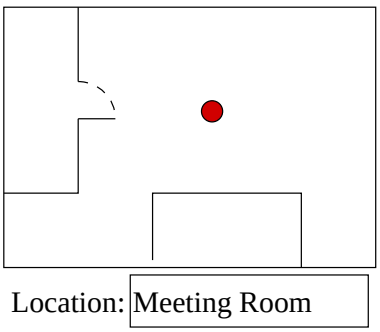


Figure 2.3: Layout of indoor navigation application

The application display the users current location on the map using a red dot. The user can enter the name of a location of interest. The application will then calculate a route to the location and display it on the map as dotted lines.

The navigation application must be supported by our system. To describe the data flow we first describe a usage scenario for the application.

Usage

The user start the application and enters a location named Meeting Room. The user navigates the building following the path depicted on Figure 2.4 with dotted lines. The mobile device will not contain any data before start up.

The route taken by the users takes the user from the current location to the location of interest marked with an x. During which the user will pass trough zones A trough D. When the user nears the locations marked by 1, 2 and 3. The application will load the portions of the map the user is likely to enter.

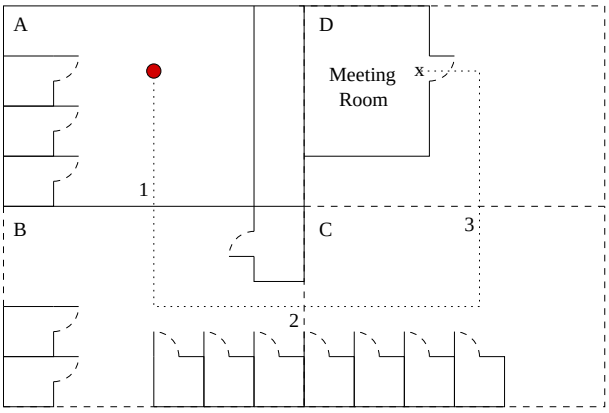


Figure 2.4: The route the user follows.

The next section will describe the data flow involved in the operation of the application.

Data Flow

Figure 2.5 on the following page shows the data flow of how the interaction between the application and the server works.

When the user starts the mobile device it do not contain any map data, because of this the application request map data for zone A, from the server which is retrieved and displayed to the user. As the user slowly moves towards the border of zone A and get closer to zone B the application will ask the server for map data for zone B. When the map data for zone B is done downloading it will be stored in the cache till the application needs the data. When the user crosses the zone A-B border the application will request the map data for zone

B, which at this point is stored locally in the cache, the cache will then provide the application with the data. When the user move towards the next border, the cycle will start over. Until the user reaches the destination point.

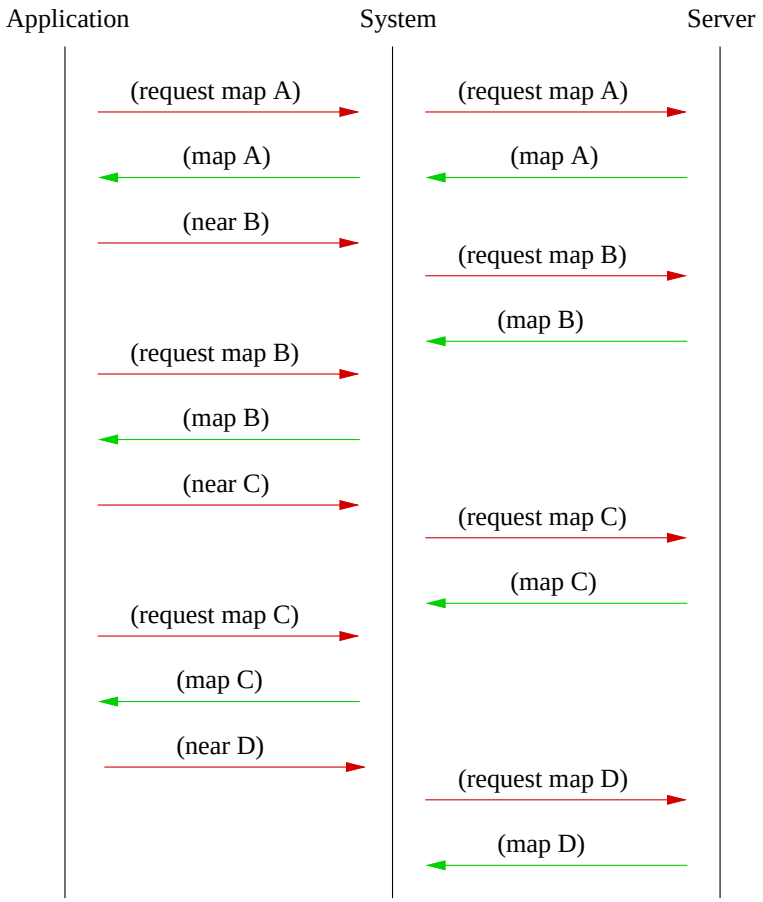


Figure 2.5: The data flow generated by the usage described above.

Requirements

Summarizing the requirements of the Indoor navigation application in relation to our system which is to facilitate easy development of this type of application.

This application requires much of the same features from our system as the digital photo album. To support this application must our system support network connections and the ability to download specific data to the cache them for later use. To support the data flow described in this use case our system also have to be able to prefetch data so it is ready when the user ask for the data.

1. Managing connection to a network
2. Delivery of image and locations data to the application

3. Managing data cache
4. Prefetching of content

2.3 Massive Multiplayer Online Game

Massive Multiplayer Online Game (MMOG) are a success on the game market and the tendency of games are to integrate more with the internet. Providing a richer game environment. MMOGs for mobile devices are still in early stages of development, but starting to surface such as the game Tibia [4].

Mobile devices have limited storage space compared to the normal computers. The internet can be used as a means to provide external data for mobile games giving the illusion of a larger storage space on the device.

Idea

Imagine a car racing game for mobile devices. The player of the game have to drive a car from the start line to the finish line, and of course be the first to cross the finish line. During the race the player will compete against other players trying to win the race as well. The player who wins the race will receive a money bonus which he can use to buy new vehicles or upgrade his existence vehicle, before the next race. Every time the player wins a race a new racing tracks will be unlocked so the player can race at this new track for an even higher bonus. Figure 2.6 display how this game could look like, the challenge for the user is to stay on the road and avoid any obstacle.

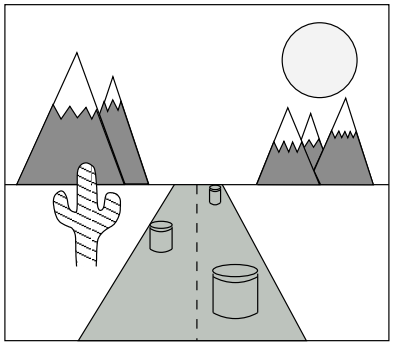


Figure 2.6: Car racing game.

Usage

The game starts with a default car the user use to compete against other players of the game. After the first race is over and the player is the winner of the race,

he decides to use his bonus to buy a new engine for his car. For the next race the user decides to play on a new race track, which has been unlocked because he won the previous run. The users on this new race track is better than the player, so he do not win the race this time and decide to exit the game.

Data Flow

Figure 2.7 show the data flow of the usage scenario described above.

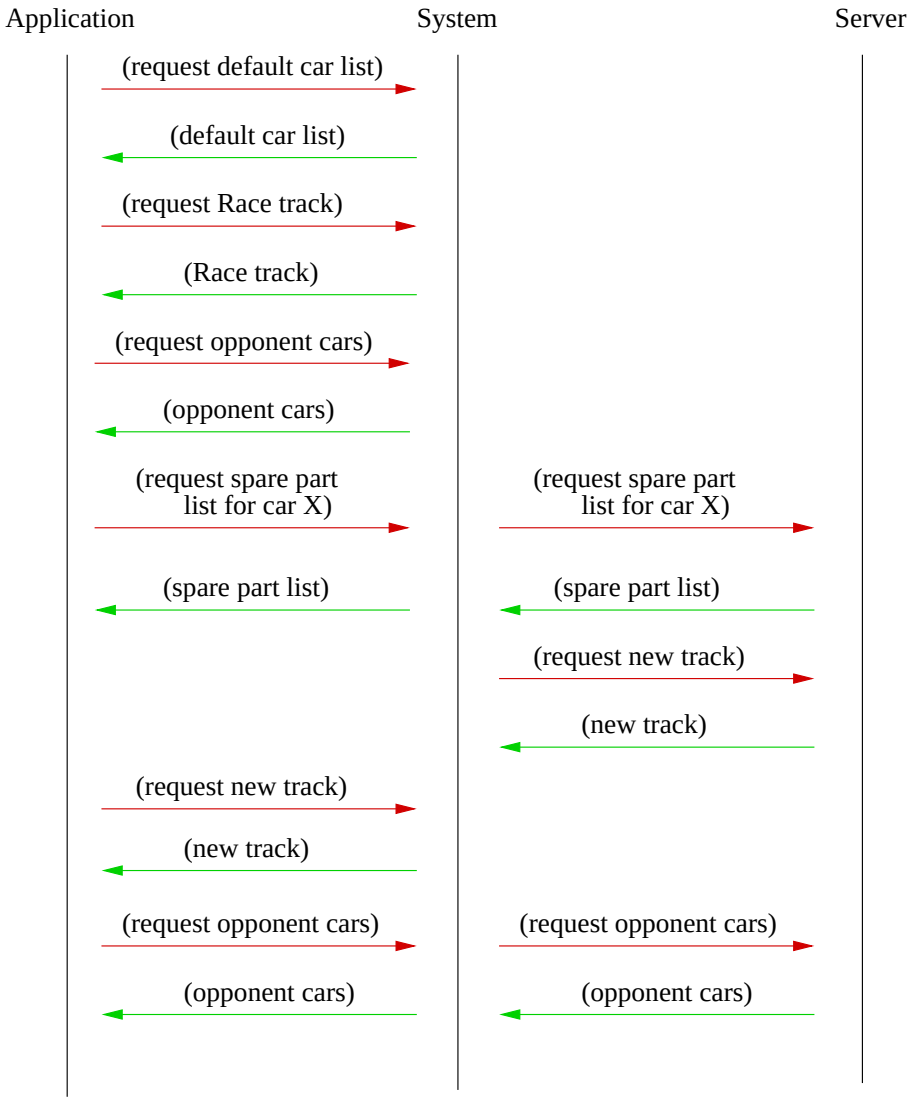


Figure 2.7: The data flow of the usage described.

At first the user have to pick the vehicle for the first race, the user have only a limited number of cars to choose from as it is his first race. To display a list of the cars possible to choose the application ask the cache for pictures of the cars.

As it is only possible to choose a few cars the application has been shipped with these cars and therefore the pictures are stored locally.

To start the race the application need to load the race track, as only one race track is possible to use at beginners, this track is also locally stored and the cache can return it after the application request it. After the race starts the graphic for the cars chosen by the other race opponent need to be downloaded to be displayed. As this is still only the first race the graphic needed to display the other cars is locally stored so the cache can reply with the needed data. After the race is over and the player has won a money bonus, he decides to buy some new spare parts for his car. To decide which parts to buy the user need a list of spare parts which would fit the car. The application then asks the cache for this list, but this list is not stored in the cache, so the cache dispatch a request for this list from the server. After the cache has received the data from the server it send the list to the application which then display the list (with pictures of all the spare parts) for the user. As the user now have unlocked a new track, the programmer of the application have decided this new track should be downloaded as soon as possible, so it will be ready when the user need it. The prefetch part of the application therefore request this track the system will then starts to download the track from the server, as it is not locally stored. After the user have bought the new car and upgraded it with a new engine he decides to start a new race on the new race track. As this track has already been downloaded it is ready for use. The user starts the run the graphic for the other vehicle need to be downloaded as they are not stored in the cache.

Requirements

The MMOG use case requires that our system supports network connection and the ability to deliver game content, over the network. It also has to support a locale store, where the MMOG can save game content to be uses later. To minimize the time the player needs to wait for game content to be downloaded, the MMOG also requires prefetching from our system. To make sure that the players keep there progress in the game, the system also needs to facilitate identification and security. This ensures that the player using an account also is the owner of the account and is not cheating.

1. Managing connection to a network
2. Delivery of games content to the application
3. Managing data cache
4. Handle user identification
5. Handle security

6. Prefetching of game content

2.4 Patching

Software development is a continuous process which is never done; there are always bugs which need to be fixed. Hence a need exist for being able to update software, often this is done by releasing patches for the consumers to download and install. New trends are moving towards automatic updating of software without user interference over the internet. By being able to change the software it is possible to ship products which is finished but still contain small errors. After the deadline it is then still possible to continue the development of the product and updating it at the users at a later time.

There exist many examples of automatic patch systems, one of the largest is the Microsoft Windows Update [5], others systems are often seen in online games where not only code but also content is patched. Other than fix errors, patch system can also be used to expand existing products with new functionality which benefits the users. This type of system is a vital part of many Massive Multiplayer Online Games where the game never ends.

Idea

Imagine a karate game for mobile devices. Figure 2.8 illustrate how such a game could look. The idea with the game is that the player is in control of a karate avatar, which has to compete against computer controlled ninjas. The more ninjas the player beat the more experience the player gets. After the player has collected a certain amount of experience the player receives a new belt (Starting with a white belt, and end up at a black belt when fully experienced). As the player receive a new belt, new combat moves are unlocked and can be used.

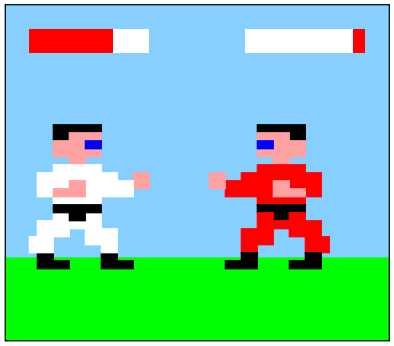


Figure 2.8: Karate game.

To describe the usefulness of patching lets imagine two scenarios; one scenario with a content update of the game in form of a patch, and the other a security

patch prevent users to exploit a bug in the game.

Usage

The content update could be introduced as the majority of players have received the black belt. To keep these players playing new content is added in the form of magic combat moves which can be learned once the black belt is obtained along with new enemies to battle.

A security update could be released as exploits are discovered. A certain combination of key presses at the right time might give players an advantage not anticipated in the game design, cause imbalances in the game. A patch for the exploit could be released and propagated to the players fixing the problem.

Dataflow

Figur 2.9 shows how the data flow for a content patch look. As the new data first will be needed when the player get the black belt, only clients who have come so far get this new content. As the server does not know which clients are using the game, the clients themselves have to ask for a new update of the game. Therefore the developer have made the game in such a way that after receiving the maximum experience points and the black belt, the game ask a server for new material for the game. If new materials exist the client starts downloading it, and when done it patch the game with the new content.

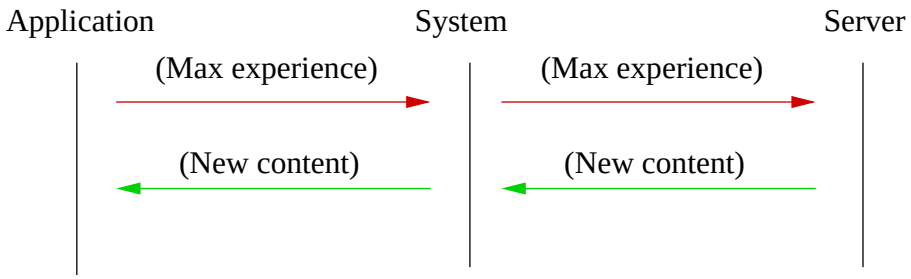


Figure 2.9: The data flow of new content.

Figure 2.10 on the next page illustrate the data flow for a security patch. The data flow for the security update looks much like the content update. The different is when the client asks for a patch. With the content patch only certain players could use the new material for the game. However with the security update all clients using the game should be updated. To make sure all clients get the security updates, the game always check if the game is up-to-date. When the game starts it connect to a remote server and check if new patches for the game are available. If this is the case, the new patch has to be downloaded and installed before the game will continue.

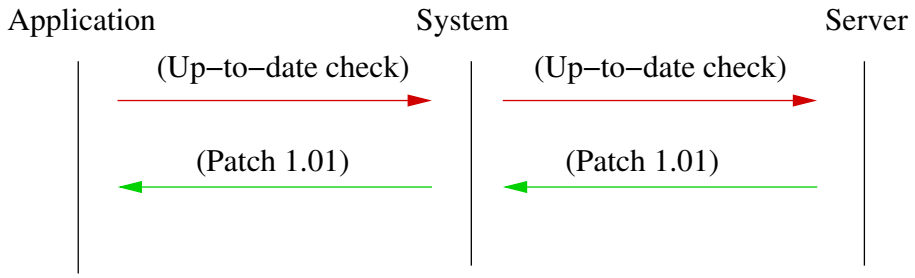


Figure 2.10: The data flow of receiving a bug fix patch.

Requirements

To facilitate the patch use case our system needs to be able to use networks communications, for downloading new patches for the system.

1. Managing connection to a network

2. Managing system updates

2.5 Feature list

This section summarizes the requirements extracted from the use cases. Additional functionality not discovered by the use cases, but deemed necessary by us is added to the feature list as well.

1. **Managing connection to a network**
   Our system must be able to establish a connection with remote servers to make use of functionality they provide.

2. **Delivery of data to the application**
   The system must be able to provide the application, with the data the application need. The origin of the data should not be a concern for the application programmer.

3. **Handle user identification**
   To make sure that the only authorized application is able to retrieve private and confidential data, the system must be able to handle identification with remote servers.

4. **Handle security**
   As some application require a secure was to send and receive data, must the system must be able to receive and send data, using secure protocols.

5. **Managing data cache**
   To save bandwidth the system should be able to locally store data for later use. As data might change over time and the mobile device has limited amount of space, the system have to be able to delete or refresh data if needed.

6. **Prefetching of application content**
   As it takes time to download data some applications might be able to predefine what data might be needed at a later time, if this is the case the system must be able to prefetch this data before the application needs it, so it will be ready when the application request the data.

7. **Managing of system updates**
   The system should support the possibility to update parts of the system. These updates could for instance be bug fixes or updating of application content.

8. **Automatic conversion of predefined data**
   This requirement is not from a use case. To save the application programmer the work of converting data to the correct format, the system should be able to recognize certain data types and automatic convert the data to the format before the application receive the data.

9. **Automatic use of the best available network and protocol**
   This requirement is not from a use case. Mobile devices might be able to use many different networks to communicate, Bluetooth, WIFI, GPRS, UMTS, etc. Some of the networks might be more costly to use than others, the system must be able to use the best network available which support the functionality needed. Beside the network the system also have to be able to use the correct protocols for specific tasks.

10. **Cache security**
    This requirement is not from a use case. The cache should be able to protect the stored data form tampering.

11. **Cache compression**
    This requirement is not from a use case. As disk space is limited on mobile devices, the cache should be able to compress the data stored.

12. **Modular design**
    This requirement is not from a use case. But as every extra feature uses both memory and disk space it should be possible to tailor the system to the individual application developers needs.

Chapter 3

Exsisting Systems

Traditional mobile software development utilizes the frameworks provided by the mobile device system platform to develop applications. On Symbian developers use the Symbian SDK, on Java enabled phones the JVM and the J2ME framework and on Windows Smart Phone and PocketPC devices the .NET Compact Framework. Although these frameworks ease the development of applications for mobile devices, users still develop higher level frameworks tailored for specific application domains to make development of applications within these domains even easier.

This chapter discusses a number of these higher level frameworks which in some form is related to the problem outlined in section 1.1 on page 6 and point out weaknesses and strengths of the frameworks. The primary purpose of the chapter is to determine if any existing systems are able to fulfil all the features identified in Section 2.5 on page 20 and defined by the problem of Section 1.1 on page 6. Failure to find such as system would mean that there is room for a new one.

3.1 GameOD

GameOD is a framework for easing the development of on-demand 3D games. The framework is developed by Frederick W.B. Li, Rynson W.H. Lau and Danny Kiis. GameOD is described in the article "GameOD: An Internet Based Game-On-Demand Framework"[6] by its developers.

In this framework all graphic elements are sent to the client when needed. Furthermore the framework is able to smoothly change between low polygon models and high polygon models depending on the avatars distance to the object. The framework is also able to synchronize animation and movement for all clients. All of the models and animations information is converted to a format, so it can be used either by the server or the clients. To minimize communication with the server the framework use caching to store date often used. To prevent high load

times for the user the framework tries to predict what data the player soon might need and starts to download this before it is needed.

The GameOD framework is specialized for 3D games, and the ability to extend the framework with new features, such as new communication protocols or security, is not an option. Again because the framework is designed for 3D games it is not possible to change some of the build-in-features, like the prefetching module. Should the prefetching be based on other criteria than the avatars view angel this framework would not be useful. The system does not facilitate any functionality to patch the framework or the 3D game running on top of it. GameOD supports the following features identified by Section 2.5 on page 20:

- **Managing connection to a network**
- **Delivery of data to the application**
- **Automatic conversion of predefined data**
- **Prefetching of application content**
- **Managing data cache**

GameOD support $\frac{5}{12}$ of the features described in Section 2.5 on page 20.

The seven features lacking from GameOD are: Modular design, managing of system updates, handle security, cache security, cache compression, handle user identification and automatic use of the best available network and protocol.

3.2 MOCA

MOCA is a framework for providing services. MOCA is presented in the article "MOCA: A Service Framework for Mobile Computing Devices" [7] written by James Beck, Alain Gefflaut and Nayeem Islam. MOCA support static and dynamic discovering of new services.

The MOCA framework consist of two parts: A registry where services are registered for later use and a notion of essential services which are services used to load other services. Examples of essential services are: Cache services, service to execute applications in a private namespace. Making it possible to run more than one application on the virtual machine. A service can also be static content used by other services, such as information or graphic elements.

MOCA only provide the basic services, all other services need to provide some sort of contents management, for the system to be able to register this service. Dynamic discovering of new services slow this system down as the framework constantly has to look for new services. Since MOCA has dynamic discovering of services, then it can change a service at run time, making it possible to update the system.

To summarize the essential functionality:

- **Managing connection to a network**
- **Delivery of data to the application**
- **Managing of system updates**
- **Managing data cache**

MOCA support $\frac{4}{12}$ of the features we deem needed by a general purpose framework. The 8 features MOCA lack is: Modular design, Automatic conversion of predefined data, Automatic use of the best available network and protocol, Handle user identification, Handle security, Cache security, Cache compression and Prefetching of application content.

3.3 CAPNET

CAPNET is a context aware middleware system for mobile multimedia applications. CAPNET is described in the article "Context-Aware Middleware for Mobile Multimedia Applications" [8] written by Oleg Davidyuk, Jukka Riekki, Ville-Mikko Rautio and Junzhao Sun.

The CAPNET middleware provides the ability to dynamically detect services which are located online or locally and move them online if the service requires resources not present on the mobile device. An example is processing power.

A special feature in CAPNET is the ability to automatic find the most suitable connection for the requirements, and seamless switch to another network type if necessary.

To support multimedia the CAPNET offer a media component, which facilitate capturing and processing of media data such as video and pictures. This component is also able to use media alarms (where speech and movement can be detected) and finally have a media storage which make the component able to store media data received from a remote server, or produced by the mobile device itself.

The CAPNET middelware is build on top of some core-components and can not be change, the core-components supports features such as media alarms, media storage, media processing and component migration. It is impossible to remove or add new core-components; it is only possible to add services. The middelware support online detection of services and remote execution of components, which contribute to the overall communication. It also uses proxies for all of it components, which make it possible to seamless move the components around. The system constantly need to be in communication with the service provider, to discover services and content that is available in the area where the mobile device is located.

To summarize the essential functionality:

- **Managing connection to a network**
- **Automatic use of the best available network and protocol**
- **Delivery of data to the application**
- **Managing data cache**
- **Managing of system updates**

CAPNET support $\frac{5}{12}$ of the features we have found needed. As it in CAPNET is possible to add new services but not update the core functionality, managing of system updates is only partially supported by CAPNET. CAPNET is missing the following features: Modular design, automatic conversion of predefined data, handle user identification, handle security, cache security, cache compression and prefetching of application content.

3.4 M-commerce

M-commerce is a framework which enabled automatic distributed service discovery for M-commerce applications. The framework is presented in the article "A Service Management Framework for M-Commerce Applications" [9] by Gary Shih and Simon S.Y. Shim.

A service could be associated to the companies ERP system, where a sale can be reported and handled. A service is provided via JINI (Java Intelligent Network Infrastructure). JINI allow for distribution of services, and also allow services to communicate individual when needed. The framework is composed of three components: A wireless service client, a service management engine and wireless services. The individual devices and interfaces to the user is handled via the wireless service client component and can be implemented trough WAP or other protocols which the mobile device supports. The wireless service client component communicate with the service management engine component, which handle security, validation of users, log events, storing of local data and guarantee atomicity of all transactions. The wireless services component is a different service the client use.

The framework purpose is primary for M-communication, and there is a great deal of security, which cannot be changed or removed completely. The framework only allow for changes by adding and removing of services. It does not allow changes to the service management engine or the wireless services component.

To summarize the essential functionality:

- **Managing connection to a network**
- **Delivery of data to the application**

- **Managing data cache**
- **Cache security**
- **Handle user identification**
- **Handle security**
- **Managing of system updates**

M-commerce feature $\frac{7}{12}$ of the features we have found of importance. Like CAPNET this framework only support managing of system updates in the form of addition and removal of services, but changes to the framework itself is not possible. M-commerce lack the following features: Modular design, Automatic use of the best available network and protocol, Automatic conversion of predefined data, Cache compression and Prefetching of application content.

3.5 SCaLaDE

SCaLaDE is a context aware middelware architecture offering information's of the users locations to other applications, which then can provide functionality/information based on the users location. It is described in the article "A mobile computing middelware for location- and context-aware internet data services" [10] written by Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli.

The system is divided into two parts: A meta data part and a service part. Meta data is composed of profiles and policies. Profiles describe the preferences of the device, such as memory, screen size and other resources the device has access to. The policies describe what has priority over other. Policies describe how the services behave, and which services have access to what. SCaLaDE also has authorization policies to make sure that only the right person can access specific services.

Services are composed of an upper level and a lower level. The upper level manages: data retrieval dependent on location and context, caching of services, so the system can run in offline mode and transaction recovery when the system losses connection.

The lower level handle the tracking of the mobile devices and trigger events like low battery, it also process the data received or send and convert it to the correct format. The low level components coordinate their operation with the policies.

To summarize the essential functionality:

- **Managing connection to a network**

- **Delivery of data to the application**
- **Automatic conversion of predefined data**
- **Managing data cache**
- **Handle user identification**
- **Managing of system updates**

SCaLaDE support $\frac{6}{12}$ of the features we find important. As the previous systems SCaLaDE also only support adding and removing of services but not changes to the middleware itself. SCaLaDE lack the following features: Modular design, automatic use of the best available network and protocol, cache compression, prefetching of application content, cache security and handle security.

3.6 PnPAP

Erkki Harjula, Mika Ylianttila, Jussi Ala-Kurikka, Jukka Riekki and Jaakko Sauvola describe in "Plug-and-play application platform: towards mobile peer-to-peer" [11] the PnPAP middelware. The PnPAP middelware is used to handle all communication with other mobile devices, by use of different P2P protocols. PnPAP is primary used for easy exchanging of P2P protocols, which is indicated by the ability to change the communications protocols at run time. This is done by using a state machine description.

To summarize the essential functionality:

- **Managing connection to a network**
- **Automatic use of the best available network and protocol**
- **Delivery of data to the application**
- **Managing of system updates**

PnPAP feature $\frac{4}{12}$ of the feature we have found importaint, PnPAP do not support updates of the system but only update of the protocols used for the communication. PnPAP lack the following feature: Modular design, Cache compression, Prefetching of application content, Cache security, Handle security, Managing data cache, Automatic conversion of predefined data and Handle user identification.

3.7 Conclusion

Section 2.5 on page 20 describe twelve features derived from our use cases which a system for mobile device application development should be able to support. It turns out that none of the existing system supports all the features. Although most of the systems support the major features such as: Managing connections to a network and delivery of data to the application. None of the system supports the modularity and flexibility we are looking for. Most system are service oriented and support modularity to some degree, but none of the system give the opportunity to alter the already existing parts of the system only modifications in the form of new services are supported.

Features such as compression of data and the use of a cache are also not supported by any system. We find this to be a serious flaw as we are dealing with mobile devices which have limited resources.

Looking at the system which has the highest fit to the feature list is the M-commerce system with coverage of almost 60%. The reason for the high coverage is due to the focus on security of the M-commerce system. The system is however still missing some vital features such as automatic conversion of predefined data which shall make development easier for the developer.

The systems which has the best coverage of the vital features is the GameOD and CAPNET system as their focus is on the data transfer of media content as well as a focus on development of multimedia applications such as games. None of these two frameworks support any form of security to prevent tampering which is important in online games or protect sensitive information.

To conclude this chapter we see a need for a new system that handle all the described features of Section 2.5 on page 20 as none of the existing systems are able to handle the task. The next chapter will introduce the design of solution to a new system we have come up with.

Part II

Developing the System

Chapter 4

Requirements

This chapter outlines the parameters and requirements for the system developed throughout this report. We will from now on refer to the system: *Framework for On-demand Delivery of Data and Automatic Patching of Code and Content* as FoDa.

On a broad scale FoDa is suppose to help mobile application developers maintain the software they sell to customers. In terms of offering valuable services such as patching of code on the mobile devices and the delivery of updated content as well as new content. All this is done over the air and totally transparent to the mobile user, meaning no user interaction required.

The system will act as a middleware layer between the mobile devices programming platform (operating system and virtual machine layers) and the applications the developers create. A lot of research has gone into the development of middleware services and what is to be understood by the term middleware. Philip A. Bernstein describes it in the article: "Middleware: A Model for Distributed System Services" [12]. He outline criteria's for middleware services one of which is that it must have implementation on multiple platforms and a transparent API meaning the API can be accessed without modification. He distinguish between middleware services and middleware, in that middleware services must be accessible remotely and middleware is the term he use for frameworks that include an API a user interface and tools to support them. Another article covering middleware services is "Dynamically Programmable and Reconfigurable Middleware Services" [13] written by Manuel Roman and Nayeem Islam. This article encourages the development of middleware services for mobile platforms to assist the development of advanced applications. Their focus is that the middleware should be dynamically upgradeable allowing application programmers to configure and upgrading the middleware with out interrupting the execution of the applications on top. Another article "Middleware Challenges Ahead" [14] written by Kurt Geihs support the idea that current middleware applications on mobile devices are to monolithic and more flexibility is needed. The ideas presented in the articles will be reflected in the design of the architecture of the system we are

to develop.

Using the feature list described in Section 2.5 on page 20 covering all the use cases. We setup the requirements for FoDa adding requirements not directly outlined in the feature list to complete the requirements.

1. Modular design for adapting to various mobile devices and easy extending and modification of the framework
2. On demand delivery of data
3. Automatic conversion of data to useable objects
4. Prefetching of application content
5. A data cache to lower bandwidth usage and optimize performance of applications
6. Automatic content and patch updates
7. Automatic detection of available network protocols
8. Automatic selection of the best network protocol
9. Automatic manage all network connections
10. Security including user identification
11. Encryption to prevent tampering of content and application data
12. Compression to conserved limited storage space
13. Low footprint, including memory consumption, CPU usage and storage space usage.
14. Platform independent architecture

The addition of the requirement for a low footprint is derived from the fact that most mobile devices have limited resources and that the system should be able to take this into consideration leaving as many resources for the application developer as possible.

The platform independence requirement stems from the idea that we want to create an architecture for as many mobile devices as possible as described by Philip A. Bernstein [12]. This means that the architecture cannot limit it self to specific platform features of the devices, although some common features are expected such as the ability to use network connections.

FoDa is intended to be used by mobile application developers with a need to maintain the applications they distribute to clients, here referring to customers

acquiring mobile software and running it on their mobile devices. The term "maintain" is not limited to the updating of the software code but also the software content used by the applications such as graphics in games etc.

The report describes a diverse set of use case scenarios (see Chapter 2 on page 9) ranging from applications which determine your position on a map based on wireless signals to massive multiplayer online games as known from the PC game industry. FoDa supports all scenarios described.

4.1 Specification

This Section describes the specification for FoDa. This is done by defining a set of criteria's which is rated by importance according to the system. The importance rating is divided into three degrees of importance: Critical, important and neutral. The importance criteria's are defined in definitions 4.1, 4.2 and 4.3. Table 4.1 on page 34 displays all the criterias and their rating. The criteria's are not meant to be measureable, although some might be, but more system philosophical criteria's intended to guide the design and planning of FoDa in terms of what functionality to give focus.

FoDa developed in this report will consist of architecture and an example implementation of the architecture.

Definition 4.1. (*Critical*)
A critical importance rating is given to criteria's that are crucial to the systems operability and functionality. This includes core functionality without which the system cannot function properly.

Definition 4.2. (*Important*)
An important importance rating is given to criteria's that are of importance to the system. Functionality which is not directly related to the core of the system, but still provides useful functionality is rated important.

Definition 4.3. (*Neutral*)
A neutral importance rating is given to criteria's of lesser or no importance to the system.

We consider the following criteria's for the system: Modularity, scalability, security and stability.

Definition 4.4. (*Modularity*)
The idea that parts of a system is viewed as independent modules which can be replaced or removed entirely without affecting other parts of the system.

Examples of systems where modularity is important are integrated developer environments like Eclipse that uses a plugin system. Often scripting languages provide a form of runtime modularity to games and applications.

In relation to our system this criteria is to make sure that the system is flexible enough. We require high modularity for reasons of limited resources on the client. Modularity can make the system compact in that unnecessary functionality is removed.

> **Definition 4.5. (*Stability*)**
> The systems ability to perform its operations flawlessly and without crashes or malfunctions.

Examples of systems where stability is important are operating systems, embedded systems, real-time systems, medical care systems etc.

In relation to our system architecture this criteria is tied to the modularity in that removing or adding modules should not compromise the stability of the system. One could argue that stability is a critical criterion for any software system and especially backbone system. Stability becomes even more important if the software is to be sold to customers. As we are creating a system as a proof of concept our main focus will be to create as much of the architecture as possible. Stability of systems is often an iterative process of fixing errors and running the system again. Development time available to us is also a factor to consider; therefore we deem this criterion important only, where as a production environment would deem this criterion to be critical.

> **Definition 4.6. (*Security*)**
> The systems ability to handle integrity of data as well as the verification of data and determine that the data is actually send from valid sources.

Examples of security are found in banking systems as well as most company infrastructure systems. Most people have tried to use a username and a password to gain access to some system.

In our system this criteria translates to the integrity of the data that is delivered to the client and way to make sure the data originates from valid sources. Also the ability to make sure that no one gets hold of data they are not suppose to hold. We deem this criteria neutral as it will not be the focus of the project as it is merely proof of concept.

> **Definition 4.7. (*Scalability*)**
> The systems ability to expand and contract as the demands on the system resources vary.

Examples of scalability is found in various distributed systems where clusters of computers can be added to expand the resources in a grid as need arises, and when no longer needed they can be removed again. The internet and DNS are both good examples.

In relation to our system architecture this criteria will add architectural constraints that enforce modularity.

Definition 4.8. (*Usability*)
Simple and easy to use interfaces and constructs, that limit the amount of work required by the programmer.

Example systems where usability is of importance is system interfacing with users like word processor, finance applications. Basically any system where humans are supports to interact with the system. In our case usability refers to the interaction between the system and the application developer. Meaning focus is on easy to use interfaces and construct and limiting the amounts of work required by the developer to produce an application. We deem this criteria important.

Criteria	Critical	Important	Neutral
Modularity	✓		
Security			✓
Scalability		✓	
Stability		✓	
Usability		✓	

Table 4.1: System criteria's rated by importance.

The system consist of a server/client configuration. Where the client is located on the mobile devices and the server is an application running on server systems providing data to the client in someway. Access to the internet is required for both the server and the client.

4.1.1 Client

The client must serve as a middleware layer for mobile device applications that need the features of the system.

The client part of the application must provide an easy to use interface for the mobile application programmers from which they are given access to the functionality provided by the middleware application in a transparent manner.

The middleware system should take into account the nature of the mobile devices, meaning accommodate and compensate for the loss of connection and consider the resources available on the client.

The footprint of the middleware should be as small as possible. As to not waste client resources that may be need for the applications developed.

The middleware must provide means of upgrading or patching applications running on top of it without client user interaction. Upgrades are delivered by the server over the air.

4.1.2 Server

The server must be able to deliver on demand data to the client. Examples of data are graphics, sounds and code.

Using a game as the example application running the system, the server could be designed to handle fitting of game content to the devices requesting the data. The server should be able to handle peak periods of high load as new patches for the game is released. The design of the server to handles these various tasks are beyond the scope of this report as the priority of the project is the client. Therefore the server will only be designed in the minimal scope needed to support the example applications of the client.

Chapter 5

Architecture

This chapter describe the architecture of the system. To start of we discuss the concept of frameworks, and what framework technology can do for the system we are developing. After this we will describe the various layering schemes which are relevant for the system and introduce the different plugin architectures relevant to FoDa.

5.1 Framework

Frameworks is a proven software technology to reduce cost and improve quality of software, their primary strength are modularity, reusability and extensibility according to the article "Object-oriented application frameworks" written by Mohamed Fayad and Douglas C. Schmidt [15].

Frameworks are designed to solve domain specific problems, it is then the application developers task to decide if a specific framework is suitable for an application. In the article *Choosing an object-oriented domain framework* [16] written by Garry Froehlich, H. James Hoover and Paul G. Sorenson, the decision process is divided into three steps:

1. **Determinant if the framework is to be immediately rejected**
   If the framework do not support any of the needed features the framework is immediately be rejected.

2. **Determine if the framework is clearly suitable**
   This is done by comparing the framework functionality with the requirements of the application.

3. **Assess the level of uncertainty**
   Determine how many of the applications requirements there is no functionality for in the framework. Many missing features result in a high level of uncertainty as the developer will have to implement the required features that are missing, which is likely to make the development time longer.

The domain of FoDa is derived from the number of use cases described in Section 2 on page 9, which resulted in a feature list. FoDa must provide functionality to cover all the features identified.

A framework works by providing specific hook methods (framework functionality) which provide a interface to the application developer. Via these hook methods the application developer can use the functionality provided by the framework. The hooks can be facilitated in a number of ways. Frameworks are classified on how they facilitate these hooks. In general two classifications for frameworks exist: White-box and Black-box frameworks.

White-box frameworks facilitate hooks by object-oriented languages features like inheritances and dynamic binding. The application developer inherits from the frameworks classes and overrides predefined hook methods.

Black-box frameworks use predefined interfaces for components, which can be plugged into the framework using patterns. Black-box framework does not require any insight into the inner workings of the framework, and usage often occur trough the use of compositions and delegations instead of inheritance. The black-box frameworks are more difficult for the developer to develop, as a very clear interface of hooks needs to be defined. This means that the framework developer must anticipate the way the users are to use the framework in a higher degree that in white-box frameworks, where the users easily can extend the framework.

Based on the requirements (see Chapter 4 on page 30) we chose to create FoDa as a black-box framework to support the usability criteria. To anticipate the needs of the user we have derived functionality from the use cases of Chapter 2 on page 9. Furthermore we give the users the possibility to add and remove hooks trough the use of a plugin architecture.

In the article *Object-oriented framework-based software development: problems and experiences* [17] written by Jan Bosch, Peter Molin, Michael Mattsson and Per Olof Bengtsson, it is encouraged to choose a maintenance strategy when designing the framework. This is due to the fact that once applications are written using the framework, changing the framework can be difficult as it might break already existing application bindings to the framework rendering the applications inoperable.

Section 5.3 on page 39 describe our choice of plugin architecture and at the same time the maintenance strategy of the framework. The next section will cover the layering of the framework architecture.

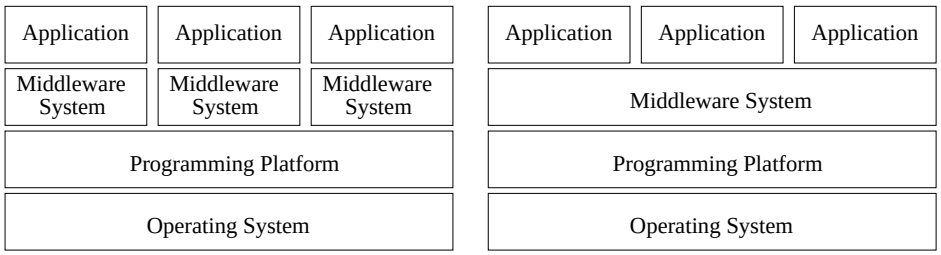
5.2 Layering

Figure 5.1 on the next page display two versions of layering we have considered for the architecture. Common for the two is that there are four layers. First layer is the operating system on which the system is suppose to run. On top of that is the programming platform layer. This layer represents any sandbox layer that

might be present on a mobile device. Examples include among others Microsoft .NET Compact Framework and Java 2 Platform, Micro Edition. On top of this programming platform lies the system itself as a middleware system and on top of this is the client applications that utilize the system.

Figure 5.1(a) display the simplest form of layering. Layers are only dependent on the ones below themselves. Specific for this type of layering is that it starts an instance of the middleware system for each application. This gives some advantages as tings are kept simple and the sandbox idea is respected in that applications are not aware of other applications running the middleware and can therefore not access data from these applications. This in turn strengthens security, which is not without importance on embedded devices. One problem with this form of layering is that resources of the devices might be consumed faster if multiple applications are running, as multiple instances of the middleware will be running with each application.

Figure 5.1(b) take a different approach to the layering. This uses shared layering. Which basically means that the middleware will only have one instance and that will service all applications. This minimizes resource usage, but it also gives the problem of handling security and separation of applications within the middleware layer, which complicates things. This form of layering is also less portable than the previous, since one cannot move an application without moving the middleware layer and all other applications using the layer.



(a) Solo layering (b) Shared layering

Figure 5.1: Architecture overview

Both of these layering scheme could be used for FoDa. We have however chosen to use the simplest layering scheme, shown at Figure 5.1(a). The cause of this decision is mainly because of two reasons; Security is a “build-in-feature” in this scheme as oppose to the other where applications share a common platform. In the simple layering scheme applications are not aware of the existence of other applications running on the same device, where it in the shared layering scheme might be possible to changes public variables of other applications unless some security measures is used.

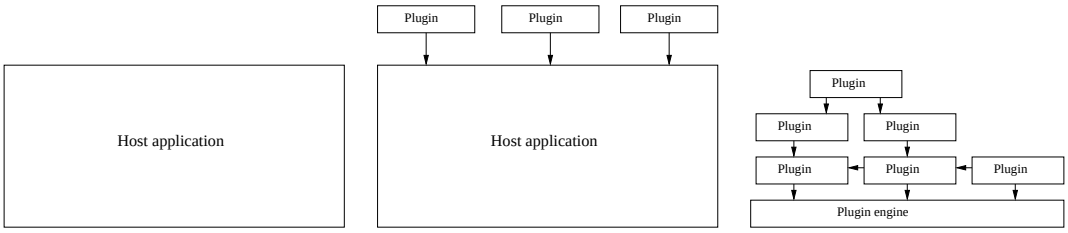
The other main reason for choosing the simple layering scheme is that by using this scheme it will be possible to run different versions of the framework for

different kind of applications, thereby make it possible to optimize our framework to the need of each application.

5.3 Plugin systems

To have the ability to change components we have chosen to base our system on a plugin architecture, where the alternative was simple *monolithic architecture*. Dorian Birsan describes in "On Plug-ins and Extensible Architectures" [18] two plugins systems, the *traditional plugin based architecture* as we know it from most applications and the *pure plugin system architecture* as seen by applications such as Eclipse.

Figure 5.3 conceptualise these three kinds of architectures.



(a) Monolithic Architecture (b) Traditional Plugin Based Architecture (c) Pure Plugin Based Architecture

Figure 5.2: Architecture overview

5.3.1 Monolithic Architecture

The monolithic architecture is actually not a plugin system but a standard application with no extendability at all. There by represented as a square box in Figure 5.2(a).

Systems using this architecture are normally designed to a specific purpose. When using this system it is often possible to describe the users needs quite precise, the user know what they need, and the needs of the user is not likely to change over time. A system using this type of architecture could for instance be an inventory management system. In this system it might be possible to add new inventory items but to add extra functionality to the system a revamp of the system is necessary. The advantages and disadvantages of this system are listed below:

Advantages

1. **No dependencies** - This system has no external dependencies as everything is contained in the same application. Therefore there is no need for dependency handling.

Disadvantages

1. **Unextendable** - It is not possible to extend the system with new functionality without redesigning the system.

5.3.2 Traditional Plugin Architecture

The traditional plugin based architecture is the extension of the monolithic architecture to support plugins. This is the type of plugin system most often seen in applications. It is possible to add new functionality within specified parameters. The architecture is illustrated in Figure 5.2(b) on the preceding page.

Most modern web browsers implement this plugin architecture. Two of the major browsers: Firefox and Internet Explorer implement it. The plugins are linked to the application trough well defined interfaces and not compiled into the application. Most plugins are only activated when the host needs them as well.

Development of applications is an ongoing process. Often as an application is released to the users, response from the usage leads to new functionality. The plugin based architecture is one way to handle this by having a core application which satisfy most needs of the users and at the same time allow users to make changes to the application as they see fit. Applications using this type of architecture often only allow for extendibility in certain areas of the application limiting the functionalty extending the application to be with in the application domain.

Advantages

1. **Extendable** - Allows for new functionality to be added within the application domain. Without redesign the application.

Disadvantages

1. **Resource usage** - The core of the application is large and unused functionality is taking up resources.

2. **Limited extendibility** - Only certain functionality is possible to add to the system without redesign of the architecture.

5.3.3 Pure Plugin Architecture

Figure 5.2(c) on the previous page display how a pure plugin architecture (dynamic architecture) could look like. The key in this architecture is the small engine which modules (plugins) can be hooked up to. This architecture is highly dynamic as it is the modules used in the system which make up the architecture.

Making a system of only modules it is possible to tailor a specific product to the needs of the costumers, even if the costumers need to use the product for different purposes.

The Eclipse Framework [19] is an example of an implementation of the pure plugin architecture.

A system using this kind of architecture could for instance be a game engine. By making the game engine of small replaceable parts it is possible to modify the engine to support different kinds of games without the need to redesign all of the architecture. If one game needs a special kind of physics it is easy to replace the physics with another and still keep all the other modules intact.

Advantages

1. **Extendable** - Possible to add new functionality to the system without the need to redesign the architecture.
2. **Meet changing requirements** - Easy to modify the system to meet changing requirements.
3. **Tailored resource usage** - Tailor the system to only use the plugins that are needed thereby saving resources.

Disadvantage

1. **Dependencies** - Dependencies among plugins can lead to complex dependency structures in the architecture.
2. **Complexity** - Lots of plugins can increase the complexity of the application.

5.4 Conclusion

Mobile software development is often characterised by its short time to market pressure. Meaning development of applications for mobile devices has a very limited time frame. This in term with the heterogeneity of the mobile devices scream for flexibility, modularity and reuse of the tools and code used to develop the mobile applications.

That is why the use of plugin architecture for the base of mobile applications is fitting as plugin architectures can facilitate flexibility, modularity and reuse to a degree not present in monolithic architectures.

Dorian Birsan describe the pure plugin architecture in contrast to the traditional plugin architecture in the article “On Plug-ins and Extensible Architectures” [18]. Here Dorian Birsan concludes that the pure plugin architecture is a powerful tool but has just recently emerged as a robust and enterprise-level

quality architecture. But there is still a number of issues which have to be dealt with in future research. This includes security, performance and support for a range of platforms, which are relevant to FoDa.

The system we are developing is a multi purpose system. This means the system has to support a wide range of applications. The use cases described in Chapter 2 in particular. To support these application an architecture which makes it easy to modify the system to the developers needs are appropriate.

The monolithic architecture does not facilitate the degree of flexibility and modularity needed by the system, so that architecture is not an option.

The plugin architecture supports to some degree the level of flexibility and modularity needed. The problem with the architecture however is that the system must support all the functionality the developer could require, which leads to a big and complex model, where in most cases application would only use a small part of the functionality provided by the system. The size of unused functionality is an argument against this architecture as mobile devices often have very limited resources such as memory and storage.

The pure plugin architecture remedy the problem with the size of the traditional plugin architecture as features unwanted or unneeded can simply be removed while keeping the flexibility and modularity at a high. The simplicity of the architecture is also a plus in that core engine only needs to contain plugin registration functions, load and unload functions. All other functionality can be added at will using plugins, making the system very receptive to developers needs. This is why our choice of architecture falls on the pure plugin architecture.

Chapter 6

Design

The previous chapter described various plugin architectures. We decided to use the pure plugin architecture for the system we are to develop during this report. In "Assessing the complexity of software architecture"[20]describes Mohsen Al-Sharif, Walter P. Bond and Turky Al-Otaiby how a central activity of designing a software architecture is decomposing the system into subsystems. They also describe how sharing of data among systems components, has influence on the complexity of the system. They conclude that the least complex way to share data is to send it directly from one component to another. In this chapter will we describe how we have divided the system into subsystems, which communicate directly with each other.

6.1 Architecture Overview

Figure 6.1 on the following page show an overview of the system architecture. To simplify the figure only the plugins that make up the backbone of the system are displayed. A more detailed description of each part of the backbone follows in their respective sections below.

The architecture is layered and consists of two layers: The application layer and the middleware layer.

The Application: The application layer is where the application using the system is running. This could for instance be one of the systems described in the use case Chapter 2.

The Middleware: Middleware usually refers to software lying in between other parts of software. In our case the middleware layer occupies a place between the underlying platform and the application layer. The middleware layer is where the developed system is running. It process requests from the application layer and replies data accordingly.

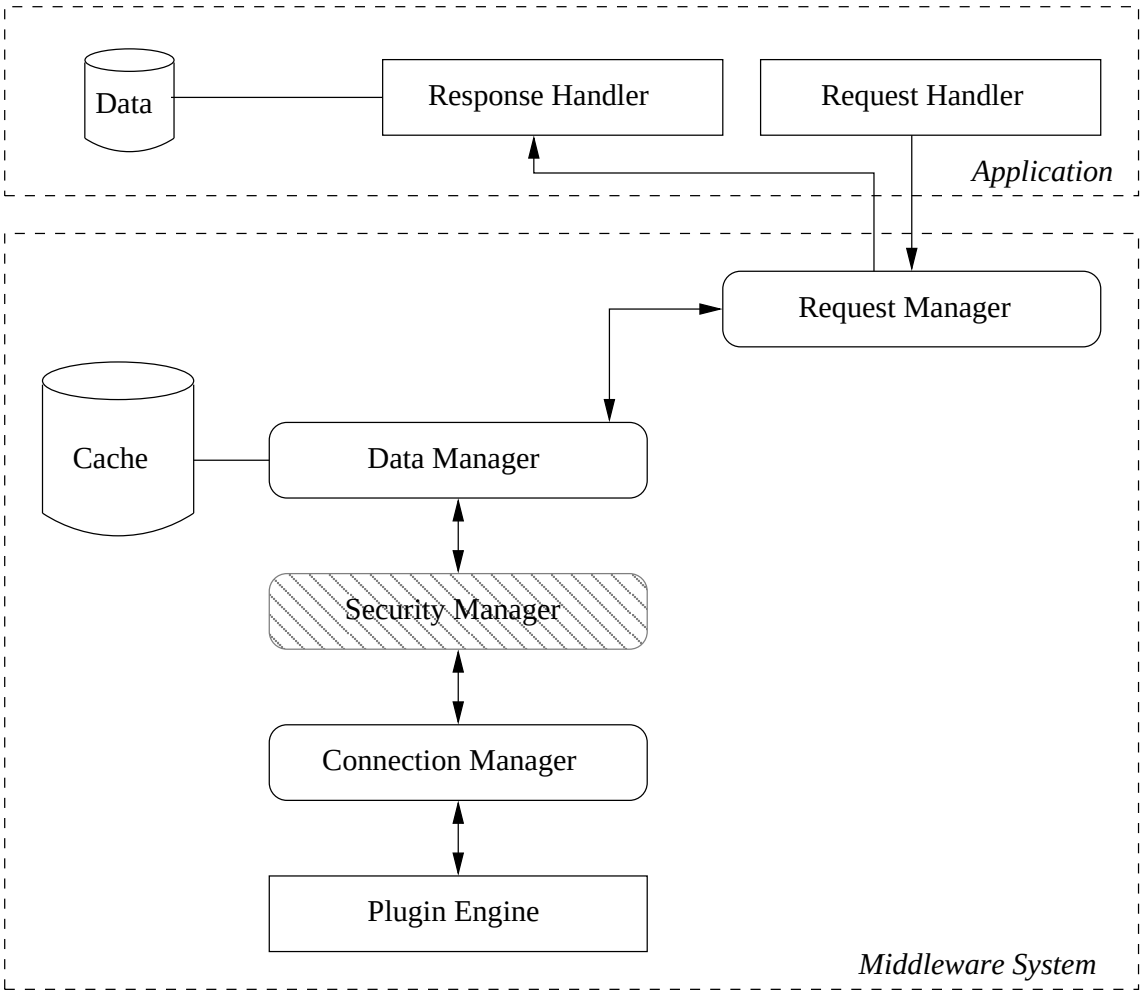


Figure 6.1: Architecture overview

The main reason for developing the system is proof of concept as described in the requirements chapter 4 on page 30. This means that commercial concerns which would normally require a great deal of attention are scaled down or even omitted. This will also be reflected in the implementation, where only the parts important to the overall function of the system according to the requirements will be implemented. The greyed out parts of the figure are parts which will not be implemented or only be implemented as a shell, meaning no real functionality is present.

6.2 The Application

For the application to make use of the middleware systems functionalities, a request for data will be needed. A request is represented on Figure 6.1 on the facing page in the form of the *Request Manager* which serve the purpose of relaying requests from the application layer to the middleware layer. Upon receiving a request the system in the middleware layer will process the request and forward a response containing the data satisfying the request to the response handler. The application will receive the data from the response handler and be able to use the newly received data.

6.3 The Middleware

The middleware system consists of several plugins each of which is responsible for a specific domain. Using the pure plugin architecture fulfils the requirement of modularity as each plugin can be replaced and new ones can be added. The following sections design each backbone plugin in detail specifying responsibilities for each plugin.

6.3.1 Request Manager

The *Request Manager* serves as the interface between the application layer and the middleware layer. It is tasked with handling requests coming from the application and relay them to the system in the middleware layer as well as replying responses to the application from the middleware system. The *Request Manager* is there after the only plugin in the system which the application communicates directly with, thereby encapsulating an interface for the application to the middleware system.

6.3.2 Data Manager

The *Data Manager* is responsible for the actual data retrieval that satisfies the request comming from the *Request Manager*. Connected to the *Data Manager* are a cache plugin and the security plugin.

Upon receiving a request from the *Request Manager* the *Data Manager* will enquire the cache plugin to satisfiy the request. If the request is satisfied the data found will be return to the *Request Manager*. If the data is not found the *Data Manager* will request the data to be downloaded by forwarding the request to the *Security Manager* which is described in detail in Section 6.3.3 on page 47. Upon receiving data from the *Security Manager* that satisfies the request. A response is forwarded to the *Request Manager*.

Omitted from Figure 6.1 on page 44 are the plugins associated with the *Data Manager*. Each of these plugins handles specific tasks related to data processing within the domain of the *Data Manager*. The detailed Figure of the *Data Manager* and all the plugins associated with it are displayed in Figure 6.3.2.

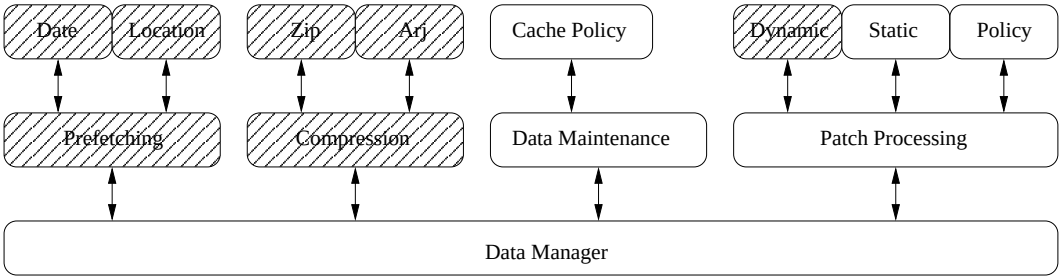


Figure 6.2: Data Manager overview

Prefetching: Sometimes it might be a good idea to prefetch certain data. For instance if a user request data element X_1 the chances that data element X_2 will be needed a little later might be high. In this case it would be possible to prefetch data element X_2 so the user do not have to wait when he is done with data element X_1 and need the next data element. Different applications might need different prefetch policies, which is why the prefecth plugin has two other plugins associated with it.

The *Prefetching: Date* plugin prefetch data based on the date of the requested data (This might be needed by a financial application, or maybe a digital photo album).

The *Prefetching: location* plugin will prefetch data based on location (the location of the avatar in a game, or the location of the user who might need map data).

Compression: By storring compressed data on the mobile client it will be possible to store more data than if the data would be uncompressed. Different approach for compressing data exist, the module associated to the *compression* describe which approach the user wish to use.

The *Compression: Zip* plugin knows how to compress and uncompress data by using the zip compression algorithm.

The *Compression: Arj* plugin knows how to compress and uncompress data by using the arj compression algorithm.

Data Maintenace: Some data is likely to change often (Like the shares of different companies stock exchange) while some other data will change

less often or never change at all (For instance the pictures in a digital photo album). Some data stored in the *Cache* should therefore be updated often to make sure the data is up-to-date while other data do not have to be updated.

The *Data Maintenace:Cache Policy* describe how long data of different kind can be stored in the *Cache* before it should be updated.

Patch processing: This plugin is responsible for the patching of the system. This means that if some code should be updated it is done by the patch processing plugin according to the update policy handled by the associated policy plugin. We divide updates into two different categories each of which are handled by separate plugins. The two categories are: Static and dynamic updates.

The *Patch processing:Dynamic* plugin is responsible for handling dynamic updates, meaning updating the application with out the need to reboot the application. This allow for updating functionality of the application while it is running. Using a game as an example dynamic updates could be useful for updating the game with new items like weapons and armor as the user encounters them.

The *Patch processing:Static* plugin is responsible for handling static updates, meaning updates that require a reboot of the application being updated. Using a game as an example, static updates are often required for updating parts of the game that is directly related to the core of the game such as the engine.

The *Patch processing:Policy* plugin is responsible for handling the policies for updates. This could be the policy describing how often a check for updates should be made by the patch processing plugin.

6.3.3 Security Manager

The *Security Manager* is responsible for handling security in the system. Primarily it is to ensure data integrity as data arrives from the data provider as well as guarantee origin of the data and verify that the data providers are who they claim to be. A detailed figure of the Security Manager and the plugins associated with it is shown on Figure 6.3.3 on the following page.

Data Integrity: During the transmission of data, some data might be corrupt or have been altered by an outsider. To make sure it is the right data the middleware provide the application with, this plugin have some data integrity plugin associated.

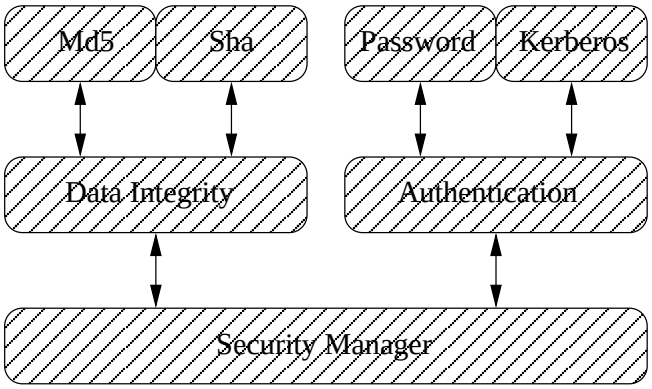


Figure 6.3: Security Manager overview

The *Data Integrity:Md5* plugin use the Md5 algorithm to check if the data recieved is the same data which have been transmitted my the server.

The *Data Integrity:Sha* plugin works in the same way as the *Md5* plugin did. The only different is that this plugin uses another algorithm to check if the data is the correct data.

Authentication: The task for this plugin is to verify that the data provider is the right data provider and not some harmful data provider. To verify the data provider is the right provider this plugin have different authentication plugin associated.

The *Authentication:Password* plugin verify that the data provider is the right provider by use of a password only known to the client and the server.

The *Authentication:Kerberos* plugin uses the Kerberos protocol [21] to authentication the connection between the client and the server.

6.3.4 Connection Manager

Before it is possible to establish a connection with a remote server it is necessary that the server and the client understand the same language. It is the responsibility of the *Connection Manager* to establish a connection between the client and the server using an appropriate protocol. Associated with the *Connection Manager* are numerous protocol plugins. A detailed figure of this is displayed on Figure 6.3.4 on the next page.

Connection Types: The responsibility of this plugin is to make sure the appropriate protocol is used to communicate with the server.

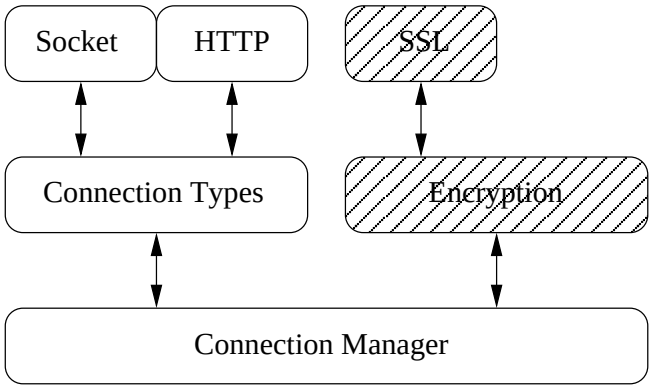


Figure 6.4: Connection Manager overview

The *Connection Types:Socket* plugin is able to establish a connection with a remote server using sockets as the communication channel. After the socket connection is establised it is possible to communicate with a remote server by sending messages over the shared socket connection.

The *Connection Types:HTTP* plugin is able to make a HTTP connection with a remote server. Communicating with the server should then be transmitted by use of the HTTP protocol.

Encryption The responsibility of the encryption plugin is to provide a means of secure communication.

The *Encryption:SSL* plugin use the Secure Socket Layer (SSL) protocol to establish a secure connection with a remote server.

This is all the plugins used in the design of FoDa, but before it is possible to implement these, it is necessary to design the plugin engine which loads all the plugins.

6.4 The Plugin Engine

This chapter describes the design of the plugin engine as it is the fundamental building block of the system developed throughout this report.

The plugin engine architecture and ideas are highly inspired by the pure plugin system described in Section 5.3 on page 39 and the Eclipse plugin architecture as described by Azad Bolour in *Notes on the Eclipse Plug-in Architecture* [22].

6.4.1 Architecture

Figure 6.5 on the next page display the class diagram of the engine.

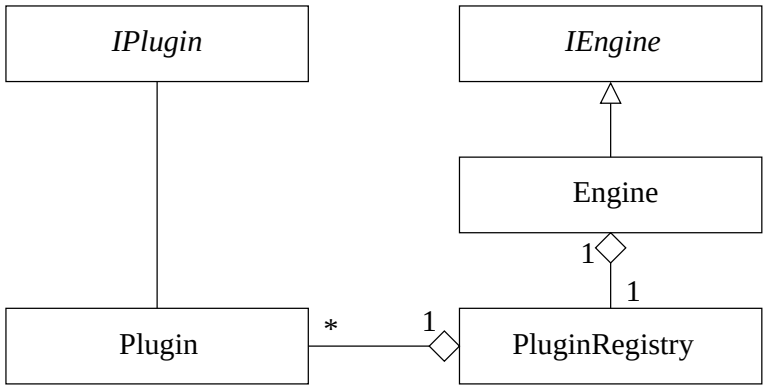


Figure 6.5: Class diagram of the plugin engine.

6.4.2 Plugin Structure

Before we describe how the engine handle plugins we will first describe what a plugin is and how it works in our system.

Basically a plugin in FoDa is just an executable program, which recive input and then is able to produce some output. Figure 6.6 shows the different parts a plugin in FoDa is made of.

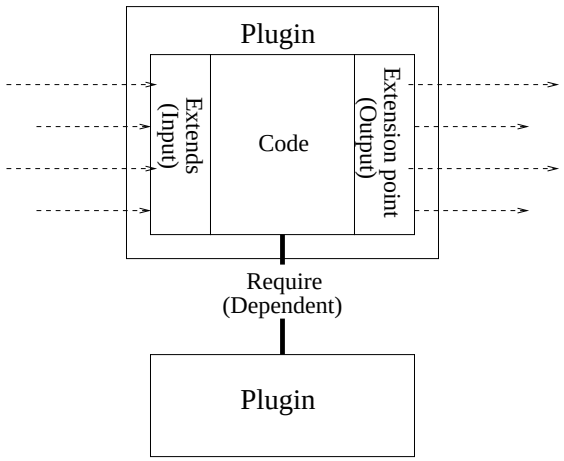


Figure 6.6: Overview of a plugin for FoDa.

Extends

To receive input from other parts of the system a plugin is able to define an extension to another plugin. By describing that plugin X should extend plugin Y, plugin X will receive input from plugin Y every time plugin Y produce the specific event plugin X extends. Section 6.4.4 on page 53 describe how to define that a plugin extends another plugin.

Extension point
If a plugin need to produce some output, this have to be done in the form of an extension point. If the developer of plugin X wants to give other developers the possibility to respond to certain output from plugin X, the developer have to make an extension point in plugin X, describing that plugin X produce output, other plugins can use. How a extension point is defined is desccribed in Section 6.4.4 on page 53.

Require
As some plugins might need a more direct communication to other plugin, it is possible to require another plugin. If the developer of plugin X require plugin Y, it is possible to make a direct reference from X to Y. Doing this gives the developer of plugin X the possibility to use methods and variables in plugin Y. Section 6.4.4 on page 53 describe how a plugin should describe a requirement to another plugin.

Code
This is where all the plugin executable code should be written. This code will only be executed if the plugin receive input from another part of the system, in form of an extension point.

Every plugin in FoDa is build by use of these input and output methods. The next section will describe how the engine handles the plugins.

6.4.3 Interfaces

Interfaces provide an entry point to the engine for the plugins. The engine contains two interfaces which are exposed to the plugin developers: *IEngine* and *IPlugin*.

IEngine

The IEngine interface exposes engine functionality to the plugin developer. Two functions are available: *GetPlugin* and *GetExtensions* as shown in Figure 6.7.

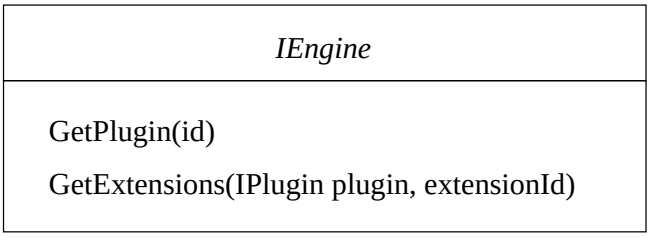


Figure 6.7: The IEngine interface.

The *GetPlugin* function can be used to direct access functionality in other plugins. Use of this function effectively creates a dependency to the plugin which is retrieved.

The *GetExtensions* are used by plugins that have extension points to retrieve all plugins which extend a certain extension point denoted by the name parameter.

IPlugin

Figure 6.8 shows the IPlugin interface, it is the interface which all plugins must implement since it identifies them as plugins to the engine. It contains one method, called run, which is only run by the engine if the plugin is a core plugin.

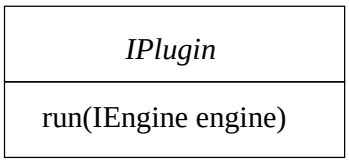


Figure 6.8: The IPlugin interface.

6.4.4 Plugin XML Language

The Plugin XML language is the language used to describe plugins enabling the engine to load them. The syntax consists of five element keywords: *Plugins*, *Plugin*, *Requires*, *Extensionpoint* and *Extends*.

```
1
2 <Plugins>
3   <Plugin filename="Plugin.dll" classname="Plugin" core="no">
4     <Requires filename="AnotherPlugin.dll" />
5     <Extensionpoint name="ExtensionOne" />
6     <Extends host="ThirdPlugin.dll" name="ExtensionThree" />
7   </Plugin>
8 </Plugins>
```

Listing 6.1: Example of the use of the Plugin XML language.

Figure 6.1 shows an example of the Plugin XML language. All plugins used in the system must be described in the engine *plugins.xml* file. This file is parsed by the engine upon start-up and plugins are loaded according to the descriptions contained in the file.

Plugin

The first element in the file is the Plugins element. This element only serves as a containment root for all plugins since a root element is required by the XML specifications [23].

The next element is the Plugin element, which constitute a plugin. It has three attributes: *filename*, *classname* and *core*.

The classname attribute is the class which implements the *IPlugin* interface, meaning it is the plugin in the dll.

The filename attribute is the name of the plugin dll file. This and the classname are also used as the unique identifier for the plugin. Uniqueness is guarantee by the underlying file system.

The core attribute is a boolean attribute which determines whether the plugin is a core plugin, meaning that the plugin will be started by the engine when loaded. More specifically it means that the engine executes the plugins run method when the engine loads the plugin.

Requires

The require element is used to describe dependencies among plugins. The element contains one attribute: *filename*.

The filename attribute is the unique identifier of the plugin on which the plugin is dependent. A plugin is dependent of another plugin if it calls methods directly on the other plugin. Listing 6.1 on the facing page shows that the plugin Plugin.dll is dependent on the plugin AnotherPlugin.dll.

Extension point

The extension point element is used to describe a point in the plugin which can be extended by other plugins. When a plugin contain an extension point it effectivley becomes a host plugin. Host plugins are responsible for calling extensions in other plugins. The extension point element has only one attribute: *name*.

The name attribute denotes the name of the extension point which serves as an identifier for other plugins.

In Listing 6.1 on the preceding page the Plugin.dll contains one extension point named ExtensionpointOne.

Extends

The extends element is used to extend other plugins functionality. The element has two attributes: *host* and *name*.

The host attribute denotes the plugin which is extended. This is the unique identifier for the plugin.

The name attribute denotes the extension point in the plugin which is extended.

In Figure 6.1 on the facing page the Plugin.dll extends the extension point ExtensionThree in the plugin ThirdPlugin.dll.

6.4.5 Events

Events can be used to pass information between plugins without creating dependencies, thus making events an effective tool that strengthens modularity and flexibility of the system. Generally events are broadcast to all subscribers resulting in a receive and react pattern. Subscribers receive an event optionally including data associated with the event and react to the event. This has the advantage of not leaving dependencies in the system, except from subscribers that are dependent on a specific event occurring. Figure 6.9 displays a broadcast event system.

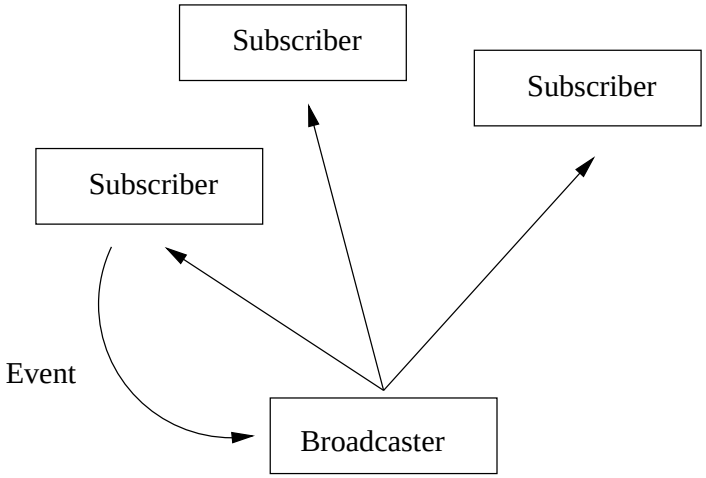


Figure 6.9: The broadcast event system.

Event systems can be made so that only events relevant to each subscriber is passed to the subscriber as shown in Figure 6.10 on the facing page. This is achieved using runtime subscription. An event filter will sort out which subscribers are to receive the event based on the runtime subscriptions.

A disadvantage of the broadcast event system and the subscription based event broadcast system, is that the dataflow is only running from the event trigger to the subscriber of the event. This means that the dataflow path cannot be manipulated by subscribers of the event. So, if one subscriber wishes to alter data from an event before it reaches the other subscribers this cannot be done. The subscriber will have to retrigger the event with the manipulated data, but at that point other subscribers would already have received the unmanipulated event data.

A system where the dataflow can be manipulated is shown on Figure 6.11 on the next page. Each subscriber now manages the subscription of other subscribers to their events and they do the task of broadcasting events themselves. This creates dependencies among subscribers for the events they need. However removing a subscriber in the middle of an event flow will disable the event for all subscribers

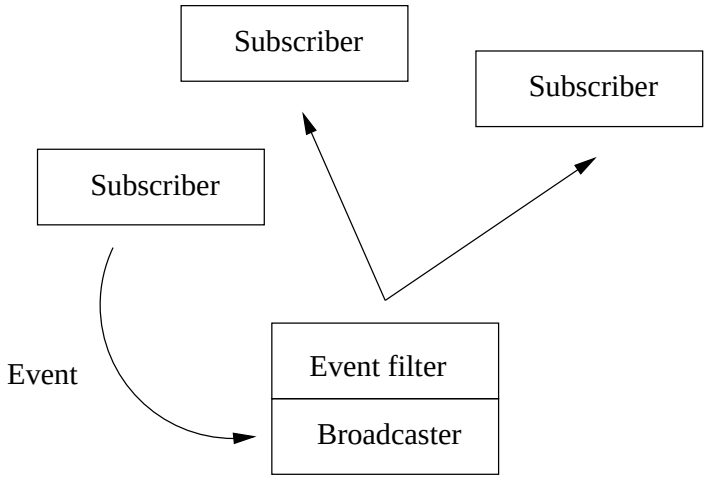


Figure 6.10: Subscription based event broadcast system

dependent on that event. An advantage of this system is that it is possible to manipulate event data, before it is passed on to subscribers above the subscriber which do the manipulation, so the data and its flow can be manipulated trough this system.

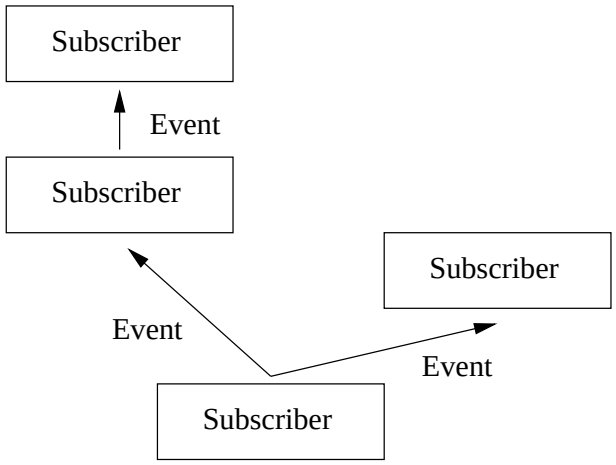


Figure 6.11: Subscription based dataflow event system.

Simulating The Event Systems

This section shows that all above event systems can be simulated by the extension point system.

The broadcast system can be simulated by letting one plugin act as the broadcaster with an event extension point which all other plugins (subscribers) must extend. Plugins can then call a trigger event function directly on the broadcaster.

The subscription based event broadcast system can be simulated by extending the broadcaster with a subscribe function which plugins can use to subscribe to specific events, making the broadcaster plugin both broadcaster and event filter in one.

The subscription based dataflow event system can be quite easily simulated by extension points. Subscribers just need to extend extension points of events they want to receive and implement the same extension point for others to extend thereby making it possible to forward events altered or not. This is the way we implement events in our system. Subscribers being the plugins, statically subscribing to events through the plugin XML file. This has the advantage of keeping the engine and plugins smaller since no code is required for keeping track of runtime subscriptions. However the event system becomes a little less flexible since subscribers cannot runtime decide whether to reveive an event or not. Nevertheless, the speed is increased since everything is only initialized once during start-up of the application as well as lowering complexity. This is preferable since we are dealing with mobile devices.

It is important to notice that the dependencies among plugins subscribing to events are not strong. This means that the plugin triggering events subscribed by another plugin can be removed without affecting the subscriber other than it will never receive the event. The subscriber plugin will still run due to the extension point system.

6.4.6 Dependency Handling

As one might have noticed in the event section above, different types of dependencies are described. In this section we describe how we handle dependencies.

The engine interface exposes the plugins that are loaded to the plugins themselves through the engine call *GetPlugin*. Using this call effectively results in a dependency in the calling plugin to the plugin it is requesting from the engine. This can result in circular dependencies. For example if plugin A calls *engine.GetPlugin(B)* and plugin B calls *engine.GetPlugin(A)*, it will result in a circular reference between A and B as shown in Figure 6.12 on the facing page.

Using Microsoft Visual Studio as the IDE the plugin A would have to have a reference to B and B would have to have a reference to A, which is not possible. The solution to the problem is to extract the interfaces of both A and B into separate dll files. Thus A implements the interface IA and B implements the interface IB. Futhermore A should have a reference to IB and B to IA. This solves the circular dependency problem, but the cost is a higher load time on applications since more dll files have to be loaded. Another way to solve the problem is to use a form of forward declaration which is possible in the .NET framework in the form of type forwarding, but it is not implemented in the Compact Framework version [24].

Using the XML file to specify dependencies means that the circular reference

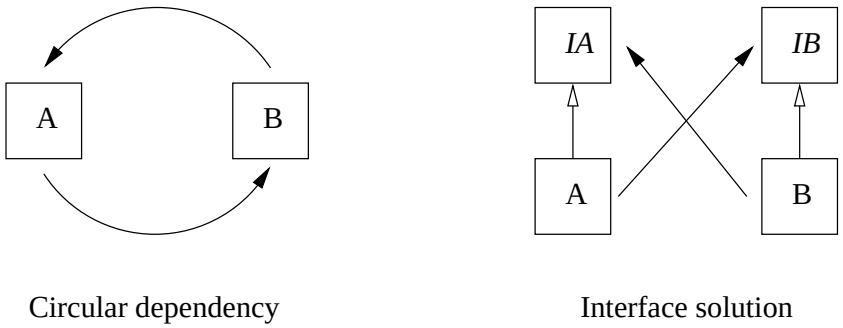


Figure 6.12: Circular reference and the interface solution.

problem is moved to the XML. In order to solve the circular problem one could make a two pass of the XML and detect the circular requires, then warn the developer that it is not allowed. Another problem is that a load order will have to be established. We do this by specifying that the plugins to be loaded first should be in the top of the XML file.

Chapter 7

Implementation

In the previous chapter we described the design of FoDa. Continuing the development process this chapter describes the implementation of FoDa. Focus of the implementation has been prioritized according to Table 4.1 on page 34. Basically this means that main focus was on getting he modularity of FoDa to work as intended and give it ability to scale. Stability of core components has also been prioritized leaving security for later. Meaning when all other features have been implemented focus will go to security.

The chapter firstly describe the choice of platform for the proof of concept implementation of FoDa. This will be followed by an overview of the system. The core of the chapter evolves around a detail description of each plugin of the system. The chapter is rounded of by an implementation status.

7.1 Platform Choice

The requirements to the system as described in Section 4 on page 30 one requirement dictates platform independence of the architecture. In this section we will decide upon a platform to implement the prototype of the system on.

Developing mobile applications, two dominant framework platforms exist: The J2ME framework and the .NET Compact framework. Both frameworks support many of the same features. But due to Microsoft strict requirements to devices running their framework applications and certification required [25]. Applications written on .Net Compact Framework are easier moved from one device to another as the framework guarantee homogeneity between devices.

The .NET Compact Framework is divided into two subsets: Pocket PC Phone Edition for PDA's with phone capabilities and Windows Mobile Smartphone for phones with PDA capabilities. This division is done to make the framework support as many capabilities as possible.

The J2ME Framework is based on the Java philosophy of write once, run everywhere. The J2ME Framework is in contrast to .NET Compact Framework

supported by a wide range of operating systems like Motorola iDEN [26], Symbian OS [27], Qualcomm Brew [28], Nokia Seris 40 [29], Palm OS [30], Wind River's VxWorks [31], Windows Mobile [32] and differt versions of Linux.

J2ME enables phones are divided into two subsets: Connected Device Configuation (CDC) and Connected Limited Device Configuration (CLDC). This is done to utilize all the capabilities of the many different devices supporting J2ME. These two versions have different J2ME implementations which are not compatible. Furthermore profiles have been introduced into J2ME which enable new features of the framework. The most widespread is the Mobile Information Device Profile (MIDP). Independent phone vendors have the possibility to introduce their own profiles with new features making devices more heterogeneous.

In practice many features of the J2ME framework is optional to implement and implementation often differ from operating system to operating system. This makes it a hazzle to get application working on the wider range of devices.

In FoDa we need to be able to load and unload classes at runtime. In J2ME this is possible by using a class loader. A custom class loader is only supported in the CDC version of J2ME, which is not as widely used as the CLDC version. The .NET Compact Framework enables the possibilities to dynamiclly load assemblies which is the equivalent of the class loader.

The choice of platform fall on .NET Compact Framework as this gives the possibility to move applications from one device to another with minimum implementation hazzle.

7.2 Plugin Overview

This section will give an overview of the implementation of FoDa. Figure 7.1 on page 61 display an overview of the system. Described in Section 6.4.4 on page 53 there are two ways of associating plugins with the framework making them part of the system. Extension points are represented as dotted arrows in the system overview figure. Requires are represented by solid arrows from one plugin to another. The plugin from which the require arrow originate is the plugin requiring the plugin the arrow points to.

Extension points can be extended with any number of extensions. A plugin is even allowed to extend its own extension point, an example of this is the Cache plugin.

Plugins can extend one another allowing for a call-back mechanism which is used to simulate events. This is widely used in FoDa. One example is the Http plugin and ConnectionManager plugin. An event is triggered in Http plugin by the ConnectionManager when the ConnectionManager requests data and an event is triggered in the ConectionManager by the Http plugin when the request is downloaded and ready for use.

The require pattern is only used in one place of the system and that is from

the Application to the RequestManager. The reason for this is that the RequestManager acts as the interface to the rest of the framework and all communication between the framework and the application pass trough here. The application plugin shown on the figure represents the application that is using the framework and it may consist of several plugins. Examples of application plugins are the Photo album, the MMO game and the Indoor Navigation application all which are described in the use cases of Chapter 2 on page 9.

The Engine is also shown to have one extension point. This extension point is somewhat special in that it has its own keyword named core in the plugin XML file. Using this plugin will tell the engine to execute the run method of the plugin upon start-up of the system. This is more thoroughly explained in Section 6.4.4 on page 52.

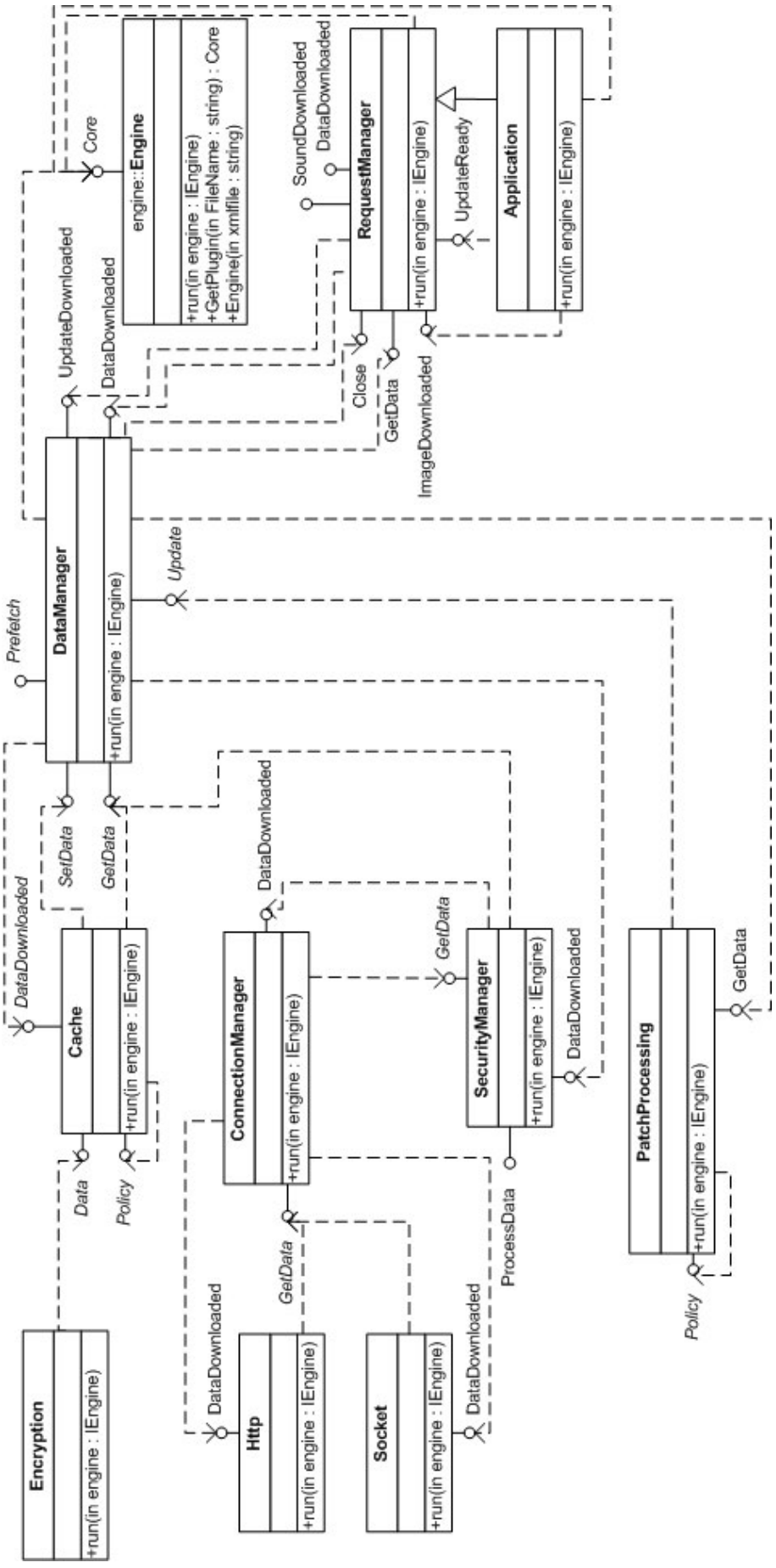


Figure 7.1: Overview of FoDa

7.2.1 Application

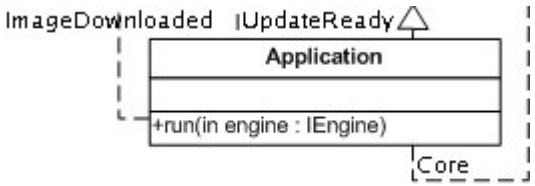


Figure 7.2: Application plugin

Figure 7.2 shows the Application plugin. The Application plugin is all the code and plugins written by the application developer using the FoDa system.

To use the system and request data the application programmer can use the interface provided by the RequestManager:

getData(filename) Use this to request data of any format. When the data is done an event will be triggered according to the type of data requested. If the data request was an image the ImageDownloaded event is triggered etc. If the data type is not recognized by the system the DataDownloaded event is triggered.

getRawData(filename) Use this if raw data is wanted. This will not be converted by the type recognition mechanism in the system, but just pass the data as it is. When the data is ready the DataDownloaded event is triggered.

Terminate() The application developer should call this when the application shut down, to notify the system that data should be persistently stored and do cleanup.

Requires: RequestManager

Applications must require the RequestManager to use the system and gain access to the interfaces.

Extends: RequestManager.ImageDownloaded

The *RequestManager.ImageDownloaded* extension is triggered when a request for an image has been satified by the system and is ready for use.

ImageDownloaded(filename, image) The filename of the ready image and the image object.

Extends: RequestManager.SoundDownloaded

The *RequestManager.SoundDownloaded* extension is triggered when a sound data has been downloaded an is ready for use.

> **SoundDownloaded(filename, sound)** The filename of the ready sound and the sound object.

Extends: RequestManager.DataDownloaded

The *RequestManager.DataDownloaded* extension is triggered when a data request has been satisfied by the system, but the system could not recognize the data or the request came from using the getRawData function of the RequestManager.

> **DataDownloaded(filename, data)** The filename of the ready data and the data object.

Extends: RequestManager.UpdateReady

The *RequestManager.UpdateReady* extension is triggered when an update has been downloaded and is ready for installation. This will notify the application developer in order for him or her to act accordingly. An example would be to display an update notification to the user of the application. The system must also be restarted for the installation to proceed this might be helpful to notify the users of whether or not to restart the application.

> **UpdateReady(filename)** Triggered when an update is ready with the filename of the update.

7.2.2 RequestManager

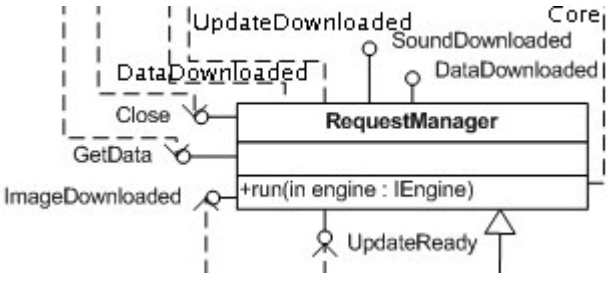


Figure 7.3: RequestManager plugin

Figure 7.3 shows the RequestManager plugin. The RequestManager plugin serve as the interface to the rest of the system. All communication between the application and the system goes trough this plugin.

Requesting data is done using the function *GetData*(*filename*) or *GetRawData*(*filename*) of the RequestManager. When the data is downloaded and ready for use an event will be triggered according to the data type. If the data is an image the ImageDownloaded event will trigger if data is sound then SoundDownloaded will trigger etc. If the system does not recognize the data format the general event DataDownloaded will be triggered and the application developer can handle the raw data.

Extension Point: ImageDownloaded

Application developers extend this extension point to receive image downloaded events.

> **ImageDownloaded(filename, image)** The filename of the image downloaded as well as the image object ready for use by the application developer.

Extension Point: SoundDownloaded

Application developers extend this extension point to receive sound downloaded events.

> **SoundDownloaded(filename, sound)** The filename of the sound downloaded as well as the sound object ready for use.

Extension Point: DataDownloaded

Application developers extend this extension point to receive unrecognized raw data downloaded events.

> **DataDownloaded(filename, data)** The filename of the data downloaded as well as the raw data.

Extension Point: UpdateReady

Application developers extend this extension point to receive notification of updates to the system and application that has been downloaded and are ready for instalment.

> **UpdateReady(filename)** The filename of the update that is ready to be installed.

Extension Point: GetData

The GetData extension point is activated when the RequestManager recieve a request from the application.

> **GetData(filename)** The filename of the data requested.

Extension Point: Close

The Close extension point is activated when the system is about to shutdown. So plugins can extends this if they need to do some clean up before shutdown, such as writing data to local disk storage etc.

Close() Called when the framework is shutting down operation.

Extends: *DataManager.UpdateDownloaded*

The *DataManager.UpdateDownloaded* is activated when the DataManager has downloaded an update. The RequestManager use this event to trigger its own UpdateReady extension point.

UpdateReady(filename) The filename of update downloaded.

Extends: *DataManager.DataDownloaded*

The DataManager.DataDownloaded is activated when data request is downloaded and ready for conversion, to the correct object type based on file format, by the RequestManager. When data is converted the extension point corresponding to the type format is triggered.

DataDownloaded(filename, data) The filename of data downloaded and the data.

7.2.3 DataManager

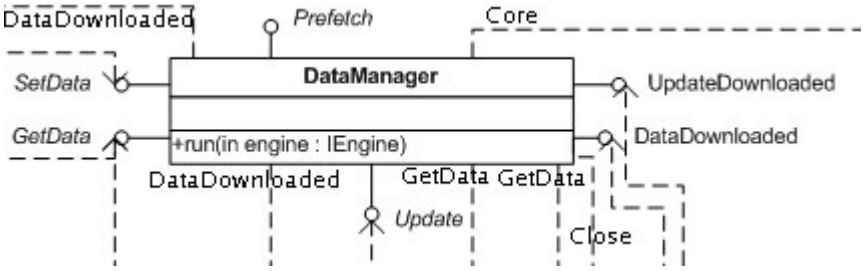


Figure 7.4: DataManager plugin

Figure 7.4 shows the DataManager plugin.

The DataManager is an administrator plugin. It is responsible for directing data requests and replies to the right places. It is the job of the data manager to first try and retrieve data from the cache and if this fails try retrieve it from the internet and make sure the cache stores the data when downloaded.

Core plugins have their run method executed upon system start-up. This means that this plugin, as the only core plugin in the system, is the central point of failure of the system and cannot be removed without making a proper replacement.

Upon receiving data from a plugin the DataManager activates its DataDownloaded extension point, which acts as an event in the parts of the system (The RequestManager) extending the extension point.

An option exist to extend the DataManager with a Prefetch plugin which are activated every time data is requested. Giving the opportunity to ask for more data than originally in the request. Effectively precaching data before it is to be used.

Extension Point: GetData

Plugins should extend this extension point if they in some ways provide a service of fetching data either from local storage or over network connections.

GetPriority() Determine the priority of the plugin based on the storage the plugin use to retrieve data. Two priorities exist: *Local* for use when fetching from local storage or *Internet* for when fetching data from the internet.

GetData(filename) The filename of the requested data.

Extension Point: SetData

The SetData extension points is activated when the DataManager has received data retrieved from an Internet priority plugin. Invocation of this extension point signals a desire to get the data stored persistently.

SetData(filename, data) The filename and data to be stored.

Close() Called when the system shutsdown, signalling to save data not yet persistently stored.

Extension Point: Prefetch

The Prefetch extension point is activated when a request for data is made, giving the opportunity to request supplemental data with the request.

PrefetchData(filename) The filename of the request made.

Extension Point: DataDownloaded

The DataDownloaded extension point is activated when data is ready to be forwareded from the DataManager to plugins which extend this extension point.

> **DataDownloaded(filename, data)** The filename of the data which has been downloaded and the data itself.

Extension Point: Update

The Update extension point is used by plugins which handle updating of the system. In our case the PatchProcessing plugin.

> **UpdateCheck()** This will get activated every time a request for data is made. It is then up to the policy of the plugin extending the Update extension point to determine if a check for updates should be made or not.
>
> **IsUpdate(filename)** This will get activated every time date is received to determine if the data is an update which should trigger an update notification for the user or is application data which should be forwarded to the request manager.

Extends: RequestManager.GetData

The *RequestManager.GetData* extension is triggered by the RequestManager, when the application using the RequestManager request data.

> **GetData(filename)** The filename of the requested data.

Extends: PatchProcessing.GetData

The *PatchProcessing.GetData* extension is triggered by the PatchProcessing plugin when it requests updates.

> **GetData(filename)** The filename of the requested update.

Extends: SecurityManager.DataDownloaded

The *SecurityManager.DataDownloaded* extension is triggered by the SecurityManager when data is ready that satisfies the DataManagers data request. The DataManager will forward the data received to the RequestManager by activating its own DataDownloaded extension point.

> **DataDownloaded(filename, data)** The filename of the data downloaded and the data itself.

Extends: Cache.DataDownloaded

The *Cache.DataDownloaded* extension is triggered by the Cache when data that satisfies the DataManagers data requests. If the Cache could not satify the data request it will trigger this event with parameters set to null, telling the DataManager to get the data from some other plugin.

DataDownloaded(filename, data) The filename of the data and the data that satisfies the request.

Extends: RequestManager.Close

The RequestManager.Close extension is triggered by the RequestManager when the application running on top of the system is shutting down. This event will activate the plugins extending the SetData extension point to save data properly before complete shutdown of the system.

Close() Tells the plugin to shutdown.

7.2.4 PatchProcessing

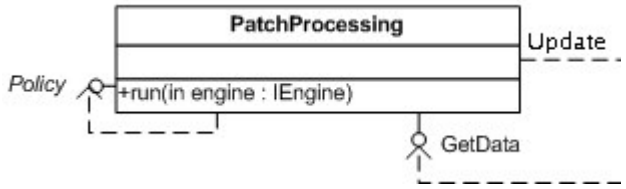


Figure 7.5: PatchProcessing plugin

Figure 7.5 shows the PatchProcessing plugin. The responsibility of the PatchProcessing plugin is to administrate the updating of the system.

The plugin initializes by reading an XML file containing entries for the parts of the system that can be updated. The entries contain information about where the parts are stored and when they were last check for a newer version available. The information is parsed into a hashtable for fast and easy access.

When the UpdateCheck extension is activated it will trigger the Policy extension point. The policy plugin will then check the hashtable for entries which according to policy require an update. This is the case if an entry is more than two days old.

Extension Point: Policy

The Policy extension point is activated when an update check is needed.

Policy(hashtable) The hashtable containing the entries that can be updated. The policy plugin should maintain the hashtable according to policy and request updates.

Extension Point: GetData

The GetData extension point is activated

Extends: DataManager.Update

The *DataManager.Update* extension is activated every time the DataManager receive a request for data or when data that satisfies a request is retrieved.

UpdateCheck() Activated when a request for data is received by the DataManager.

IsUpdated(filename) The filename to check if it is an update. Activated when data that satisfies a request is received by the DataManager.

7.2.5 Cache

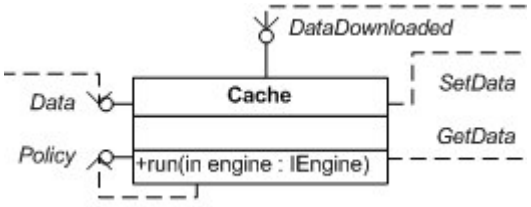


Figure 7.6: Cache plugin

Figure 7.6 shows the Cache plugin. The Cache plugin saves the data it receives locally on the device, so it can be retrieved later without the need to use the Internet. It also makes sure that data saved locally is up to date with the data on the Internet.

On initialization of the Cache an XML file containing the entries currently in the cache is read. An entry contains information about name of the data stored in the entry, the age of the data and a last read timestamp. Entries in the XML file are parsed to a hashtable, which is maintained by the Cache plugin. Upon termination of the Cache plugin the Close methods need to be executed as this will save the updated entries of the hashtable to the XML file on the local disk.

On new data received the Cache stores the data on the local disk and updates the hashtable accordingly.

On data requested the cache looks up the name of the requested data in the hashtable if the name does not exist the Cache returns an empty reply to the one

requesting data. If the data do exist it is read from the disk and the hashtable entry has its last read timestamp updated and returns the data to the requester.

Purging the cache is controlled by policies. A policy describes when data is too old and must be updated or deleted. The Cache plugin implements an extension point named Policy for this purpose. A default policy is provided with the framework. The policy simply deletes all data with an age of more than five days. The policy is implemented within the Cache plugin itself as an extension to the Policy extension point.

Extension Point: Data

This extension point gives the possibility to change data before it is saved to the local storage. This extension point could be extended to encrypt and compress data before it is stored. The following interface functions are associated with the extension point:

DataTo(data) Before data is saved to the local disk, this function is called.

DataFrom(data) After data is read from the disk, this function os called.

Extension Point: Policy

This extension point gives the possibility to add and remove policies for maintaining the cache. The following interface functions are associated with the extension point:

Policy(hashtable) The hashtable containing data entries in the cache. Apply the policy by altering the hashtable accordingly.

Extension Point: DataDownloaded

The data downloaded extension point is used to signal that the cache has retrieved data. This extension point is basically an event that is triggered every time the cache has fetched data requested.

DataDownloaded(filename, data) Event function called whenever data is ready, meaning a request has been fetched by the cache. The filename of the data and the data is passed with the event. If the data requested was not in the cache null is passed as filename and data.

Extends: *DataManager.SetData*

The *DataManager.SetData* describe the interface which the Cache must implement to extend the *DataManager.SetData* extension point. The following functions are contained in the interface and must be implemented by the Cache:

SetData(filename, data) The DataManager calls SetData with the filename and data to be stored. The plugin should store the data using the filename as an index.

Close() DataManager calls Close when about to shutdown at which point the Cache run all the extensions to its Policy extension point on the maintained hashtable and upon completion stores the hashtable in the XML file.

Extends: *DataManager.GetData*

The *DataManager.GetData* descibe the interface which the Cache must implement to extend the *DataManager.GetData* extension point. Functions of the interface are described below:

GetPriority() Determine the priority of the extension to the DataManager. The lower the number, the higher priority.

GetFile(filename) Called by the DataManager to request data from the cache. The filename specifies the file requested. This should update the Cache entry hashtable accordingly, modifying last read timestamp.

Extends: *Cache.Policy*

To implement a default cache policy the Cache extends its own Policy extension point. Default policy being anything with an age above five days is deleted from the cache.

Policy(hashtable) Modifies the hashtable by removing entries older than five days.

7.2.6 SecurityManager

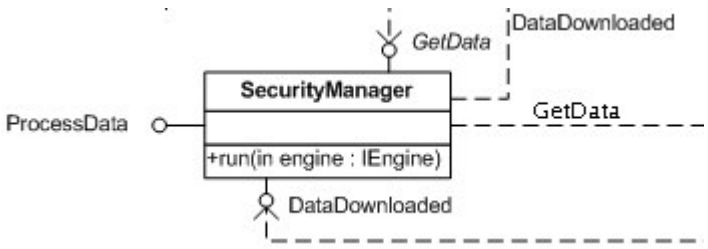


Figure 7.7: SecurityManager plugin

Figure 7.7 shows the SecurityManager plugin. The SecurityManager plugin acts as an administrator for plugins handling security, covering topics ranging

from encryption, decryption to ensuring data integrity by the use of MD5, SHA or other protocols. It also handles the order of applying the plugins to the data as the order is important. An example is doing an MD5 check on encrypted data does not yield the same result as doing an MD5 check on unencrypted data. To decide the order the SecurityManager uses the GetPriority function.

The SecurityManager receive data requests from the DataManager and process these requests to become secure, before forwarding the request to the ConnectionManager. When the data is received by the ConnectionManager the SecurityManager is notified and security plugins gets to process the downloaded data before it is passed to the DataManager and the rest of the system.

Extension Point: GetData

The GetData extension point is activated when the SecurityManager receives a request for data and has validated the request according to security policies.

GetData(filename) The filename of the data to be downloaded.

Extension Point: ProcessData

The ProcessData extension point is invoked when the SecurityManager receive a request for some data or the SecurityManager has been notified of data downloaded and ready. This extension point gives all security plugins the opportunity to process the data before it is released to the rest of the system.

GetPriority() Determine the order of execution of security plugins.

InputData(filename, data) Invoked when data is downloaded and ready to be processed. The filename of the data and the downloaded data is the parameters.

Outputdata(filename) Invoked before a request for data is relayed to the ConnectionManager. The parameter is the filename of the data requested. Enables security pluings to process the request before it is relayed.

Extension Point: DataDownloaded

The DataDownloaded extension point is activated when data has been downloaded and all security plugins are done processing the data.

DataDownloaded(filename, data) The filename of the downloaded data as well as the processed data.

Extends: *DataManager. GetData*

The *DataManager.GetData* extension allows the SecurityManager to receive requests from the DataManager which is processed before invoking the SecurityManagers own GetData extension point.

GetPriority() Determines the priority of the plugin.

GetData(filename) The filename of the data requested by the DataManager.

Extends: *ConnectionManager.DataDownloaded*

The *ConnectionManager.DataDownloaded* extension is used to receive data downloaded event notifications from the ConnectionManager. Upon receiving an event the SecurityManager will lets security plugins process the data before invoking its own DataDownloaded extension point.

DataDownloaded(filename, data) When the SecurityManager receive a "DataDownloaded()" event from the ConnectionManager it will translate the data into usable data and invoke its own "DataDownloaded" extension point.

7.2.7 Encryption

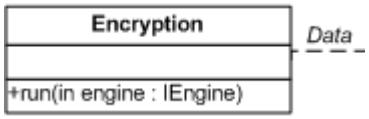


Figure 7.8: Encryption plugin

Figure 7.8 shows the Encryption plugin. The purpose of the Encryption plugin is to provide for encryption and decryption of data to and from the cache. The plugin use a simple XOR encryption scheme. The reason for encryption data in the cache is to prevent the user from tampering with the data in the cache.

Every time the Cache plugin write data to the local disk the Encryption plugin will encrypt the data before it is stored. When data is retrieved from the local disk the Encryption plugin will decrypt the data before it will be used by the system.

Extends: Cache.Data

The *Cache.Data* extension is triggered when data is read or written to and from the cache.

DataTo(data) The data to be stored in the cache. The XOR encryption scheme is applied here.

DataFrom(data) The data read from the cache. The XOE decryption scheme is applied here.

7.2.8 ConnectionManager

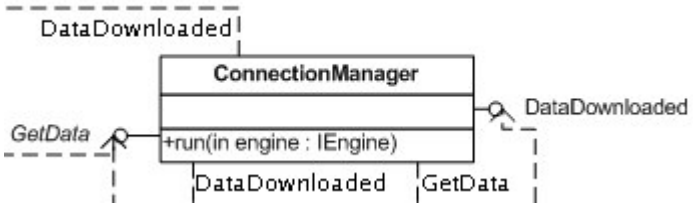


Figure 7.9: ConnectionManager plugin

Figure 7.9 shows the ConnectionManager plugin. The ConnectionManager plugin is an administrator for Internet connections. This means that several plugins can extend the ConnectionManager to enable support for multiple connection protocols in the system.

The task of the ConnectionManager is to find the best suited protocol plugin. This is done by checking the priority of the plugin as well as testing if the plugin actually establishes a connection. The ConnectionManager will then use the plugin with the lowest priority that have passed the connection test to download requested data.

Extension Point: GetData

The GetData extension points are activated every time the ConnectionManager receive a request for data from the SecurityManager. Forwarding the request to the plugin, chosen as the best suited, for downloading of data.

GetPriority() Determine the priority of the plugin. The lower number the higher priority.

GetData(filename) The filename of the data requested to be downloaded.

TestConnection() Determines if the plugin can establish a connection.

Extension Point: DataDownloaded

The DataDownloaded extension point is activated when the ConnectionManager receives DataDownloaded events from the protocol plugins which have downloaded requested data.

DataDownloaded(filename, data) The filename of the data downloaded and the data itself. Invoked when data is ready to be used.

Extends: *SecurityManager.GetData*

Extension to the SecurityManager.GetData extension point. Called by the SecurityMangager to request data for download. The Interface of the extension point is described below:

GetData(filename) Called by the SecurityManager to request the data of the filename specified.

Extends: *Socket.DataDownloaded*

The *Socket.DataDownloaded* extension works as an event that is triggeret when the Socket plugin has downloaded data. Upon receiving the event the ConnectionManager will activate all plugins extending its own DataDownloaded extension point.

DataDownloaded(filename, data) Upon data downloaded by the Socket plugin it will trigger this function. With the filename of the data downloaded and the data.

Extends: *Http.DataDownloaded*

The *Http.DataDownloaded* extension works as an event that is triggeret when the Http plugin has downloaded data. Upon receiving the event the ConnectionManager will activate all plugins extending its own DataDownloaded extension point.

DataDownloaded(filename, data) Upon data downloaded by the Http plugin it will trigger this function. With the filename of the data downloaded and the data.

7.2.9 Http

Figure 7.10 on the next page shows the Http plugin. The Http plugin is an extension to the ConnectionManager providing the capability to retrieve data by use of the HTTP protocol.

Upon receiving a request for data from the ConnectionManager the Http plugin establish a http connection to a specific server and request the data. When the data is downloaded the extension point DataDownloaded is invoked to notify all plugins extending this extension point of the new data.

All connections are none blocking as request could time out or fail or simply be slow. To achieve this, the Http plugin starts a new tread which handles the downloading, and messages the plugin when data is ready.

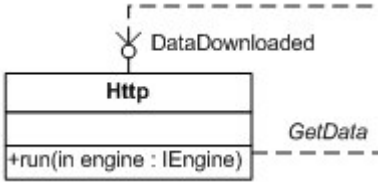


Figure 7.10: Http plugin

Extension Point: DataDownloaded

The DataDownloaded extension point is triggered when data downloaded is completed.

> **DataDownloaded(filename, data)** The filename of the data downloaded and the data object. the data.

Extends: ConnectionManager.GetData

The *ConnectionManager.GetData* extension is activated when a request for data is made by the ConnectionManager.

> **GetPriority()** Determine the priority of the plugin
>
> **GetFile(filename)** The filename of the data requested.
>
> **TestConnection()** Test if the plugin can establish a connection and therefore can be used to download data.

7.2.10 Socket

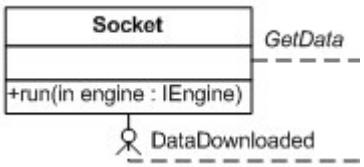


Figure 7.11: Socket plugin

Figure 7.11 shows the Socket plugin. The Socket plugin work in the same way as the Http plugin (See Section 7.2.9 on the previous page). The difference being that the Socket plugin establish pure socket connections. All connections are handled in a separate thread as in the Http plugins as well effectively making this plugin non blocking as well. The protocol used by the plugin is the TCP protocol.

This is to ensure reliable data transfer as mobile connection has a tendency to disappear. Using TCP gives a form of failure tolerance as it is guaranteed that the traffic will reach the socket at some point unless the connection is completely broken.

Extension Point: DataDownloaded

The DataDownloaded extension point is activated when a request for data is satisfied and the data is completely downloaded and ready for use.

DataDownloaded(filename, data) The filename of the downloaded data and the data object.

Extends: ConnectionManager.GetData

The *ConnectionManager.GetData* extension is triggered when the ConnectionManager receives requests for data.

GetPriority() Determine the priority of the plugin.

GetData(filename) The filename of the requested data. This will start the downloading of the data in a separate tread.

TestConnection() Test if a connection can be established enabling the plugin as a candidate for downloading data.

7.2.11 Implementation status

This section describes the implementation status of FoDa as described in the design section 6 on page 43.

Implementation decisions of what to implement is based on the criteria rating of table 4.1 on page 34. As modularity was of critical importance we have implemented the entire engine. Making it possible to create plugins that use the full feature set of extension points, requires and extends.

The DataManager is fully implemented as well as a simple cache, the ConnectionManager, PatchProcessing and the HTTP plugin.

The RequestManager only the basic interface functions needed for the use case applications to function have been implemented.

Security was of neutral importance, but we have implemented the SecurityManager with the basic features needed for example security plugins to function.

The parts of the implementation which is only partially implementet or entirely omitted is described below:

Encryption in the Cache
The Cache is encrypted, but it is with the very simple XOR encryption algorithm. This should be changed to a more advanced algorithm if FoDa were to be used commercially, but for proof of concept purpose XOR will suffice.

Security in connections
Security on communication is not implemented but support for security is present. To test if the support worked we hooked the encryption plugin to the SecurityManager and XOR'ed all communication which works.

Support for Socket communication
The Socket protocol is implemented but suffer from problems of unreliability during data transfer.

Application initiated updates
It is not possible for the application developer to initiate a specific update trough the use of a method call. An example of an application which could make use of this feature is the karate game describe in Section 2.4 on page 18. The only way to update the system is trough the automated PatchProcessing plugin.

Advanced features of RequestManager
Only the basic features of the RequestManager have been implemented. Resulting in features like convert received data to sound objects, and sending and receiving of raw data are not implemented.

Selection of best network protocol
The implementation of network handling does not choose the best protocol depending on actual measurements of the networks. Currently HTTP is prefered.

7.3 Platform Limitations

This section describes some of the limitations with choosing the .NET Compact Framework as the platform for our implementation of the system.

7.3.1 Unloading of DLLs

Unloading of DLLs or assemblies cannot be done in the .NET Compact Framework. The decision not to support this is done by the .NET Compact Framework team at Microsoft. The reason is noted on Jason Zander's Weblog [33]. Jason Zander is General Manager for the .NET Framework (DevFX) team in the Developer Division at Microsoft.

The reason stated is that the amount of administration required by the framework to support this would take up to much of the device resources and be to expensive.

> "(...) tracking is handled today around an app domain boundary. Tracking it at the assembly level becomes quite expensive." - Jason Zander

This decision effectively limits the dynamic unloading of pluins from the engine. Resulting in a situation where runtime automatic dynamic code patching is impossible. The solution to the problem is to restart the application thereby unloading and loading the dlls again. But as we shall see in the next section this is not easy to do in the .NET Compact Framework either.

7.3.2 Automatic Restart of Applications

Restarting applications is not directly supported in the .NET Compact Framework 1.0, but will be introduced in 2.0. To work around a call to the unmanaged CreateProcess function can be used. Listing 7.1 display the declaration of the CreateProcess function.

```
1  [DllImport("coredll.dll")]
2  extern static int CreateProcess(
3    String imageName,
4    String cmdLine,
5    IntPtr lpProcessAttributes,
6    IntPtr lpThreadAttributes,
7    Int32 boolInheritHandles,
8    Int32 dwCreationFlags,
9    IntPtr lpEnvironment,
10   IntPtr lpszCurrentDir,
11   byte[] si,
12   ProcessInfo pi);
```

Listing 7.1: Importing CreateProcess from coredll.dll.

To make it easier to use the CreateProcess function a wrapper can be made as displayed on Listing 7.2 on the following page.

The code that calls the CreateProcess wrapper to restart the application is displayed on Listing 7.3 on the next page.

The reboot code works in the emulator but unfortunately not on either of the devices we had access to.

7.3.3 Dynamic Instantiation of Classes

Loading plugins dynamically means that we have to instantiate the plugins at runtime. This can be done in the .NET Compact Framework using the Activator

```
1  public class ProcessInfo {
2    public int Process;
3    public int Thread;
4    public int ProcessID;
5    public int ThreadID;
6  }
7
8  private bool CreateProcess( String ExeName, String CmdLine, ProcessInfo
       pi ) {
9    if ( pi == null ) pi = new ProcessInfo();
10
11   byte[] si = new byte[128];
12   return CreateProcess(ExeName, CmdLine, IntPtr.Zero, IntPtr.Zero, 0, 0,
         IntPtr.Zero, IntPtr.Zero, si, pi) != 0;
13 }
```

Listing 7.2: CreateProcess wrapper

```
1  public void Reboot() {
2    String path = Path.GetDirectoryName(Assembly.GetExecutingAssembly().
       GetName().CodeBase);
3    CreateProcess(path + "\\engine.exe", "", null);
4  }
```

Listing 7.3: Reboot method for restarting the application.

class. This class has a static method called CreateInstance(Type) which in term will instantiate a class of Type using the default constructor.

Problem with this is that the normal way of coding applications often involves allocating resources in the constructor, but when using our system, coding pattern has changed to become event based. This means there is one call to the request manager which asks for content and when the content is ready an event is triggered. Now it would make sense to make the call to the request manager in the constructor keeping the coding pattern as close to normal as possible. This is however not possible as a reference to the engine is needed which cannot be passed to the constructor as only the default constructor is called by the Activator.CreateInstance method. Listing 7.4 on the facing page illustrate the problem in code.

Looking at the reference API manual for the .NET Compact Framework only the default constructor is available where as in .NET Framework constructors which takes an array of parameters are available.

```
1  public class APlugin : IPlugin
2  {
3    IEngine engine = null;
4    IRequestManager manager = null;
5
6    public APlugin()
7    {
8      // We cannot initialize engine or manager variables here
9      // as we have no engine reference yet. Hence we cannot
10     // request content at this point.
11   }
12
13   public void run(IEngine engine)
14   {
15     this.engine = engine;
16     manager = (IRequestManager)engine.GetPlugin(``RequestManager'');
17
18     // Now we can request content.
19     manager.GetImage(``image.png'');
20
21   }
22
23   public void IImageDownloaded(String filename, Image image)
24   {
25     // Initialize the image variables used in the project here.
26   }
27
28 }
```

Listing 7.4: Plugins are initialized using the default constructor.

Part III

Framework Evaluation

Chapter 8

Evaluation

This chapter focus on two requirements from the requirements described in Section 4 on page 30: *Ease of use* and *performance*.

The reason we focus on these is because the rest of the requirements are fulfilled by the design of FoDa. *Ease of use* stems from the usability requirement, defined in Section 4.1 on page 32. *Performance* is a combination of scalability, stability criterias and the requirement: Low footprint, including memory consumption, CPU usage and storage space usage.

We have rated security neutral, which is the reason the security of the system will not be tested. Evaluation of ease of use is done by implementing three use case applications using FoDa and record the number of code lines required by the developer to write, comparing them with the features gained from using FoDa.

One could argue that lines of code are not equivalent with ease of use. To remedy this, a tutorial is written, which illustrate the simplicity of the use of FoDa in a few simple steps. Another point is that implementations only make use of simple code construct often only one command at each line.

To further illustrate the ease of use, we port an existing game not written by us to use FoDa and record the code modifications required. We then compare these lines with the features gained.

The applications we implement are: A Photoalbum, Indoor Navigation, Jumping Game and the Port of Pocket 1945. These applications are equivalent with the use case applications, described in Chapter 2, with the exception of the massive multiplayer online game which we do not have the time to create. The Jumping game is a replacement.

The performance part of the evaluation focuses on the following criteria:

Memory usage: The amount of memory used when running FoDa the application on top.

Memory peak: The amount of memory used at peak performance.

Sytem size: Storage space occupied.

Startup time: Time it takes to start up the framework and the application on top.

Execution time: Time from when data is requested until it is ready for use.

8.1 Ease of use

Each application described in the *Ease of use* section follows the following pattern: Introduction of the application, relevant code related to use of FoDa for implementation of the application and comparison of code lines are used to interact with FoDa compared to the rest of the code of the application.

Before we describe how the application uses FoDa we outline the steps which an application developer has to follow to make use of FoDa.

8.1.1 Plugin Tutorial

This section describes how FoDa should be used to function in an application. Since FoDa is highly event driven it has not been possible to make the system 100% transparent to the user (the programmers who use FoDa for their system). This is why of this the user has to follow a few guidelines.

Listing 8.1 shows how an empty project, using the features provided by FoDa, look like.

```
1  public class Photoalbum : IPlugin, IImageDownloaded, IUpdateDownloaded
2  {
3    private IEngine.IEngine engine;
4    private RequestManager.IRequestManager manager;
5
6    public void run(IEngine.IEngine engine)
7    {
8      this.engine = engine;
9      this.manager = (RequestManager.IRequestManager)engine.GetPlugin("
           RequestManager.dll");
10   }
11
12   public void ImageDownloaded(String fileName, Bitmap data)
13   {
14
15   }
16
17   public void UpdateReady(String update)
18   {
19
20   }
21 }
```

Listing 8.1: An empty project

Describing the empty project at Listing 8.1, the first thing to notice is the interfaces which a project using FoDa should implement.

1. **Implement the "IPlugin" interface**
   The first interface, "IPlugin", should always be implemented, this interface contain one method. This method is the "public void run(IEngine.IEngine engine)" implemented at line 6, which provides the application with a main activation point. This is comparable to the "public static void Main()".

2. **Implement the "IImageDownloaded" interface**
   The next interface which the empty project at Listing 8.1 implements is the "IImageDownloaded", this interface should be implemented if the user will be requesting images. When the user request an image by use of the "getData()" function (located in the *RequestManager*) the method "public void ImageDownloaded(String fileName, Bitmap data)" will be invoked when the data is ready to be used. If the user needs to request non image data, another interface should be implemented as well.

3. **Implement the "ISoundDownloaded" interface**
   If the user needs to request sound data the interface "ISoundDownloaded" should be used.

4. **Implement the "IDataDownloaded" interface**
   If any other data is needed the interface "IDataDownloaded" has to be implemented.

5. **Implement the "IUpdateDownloaded" interface**
   The last interface the empty project implement is the "IUpdateDownloaded" which have to be implemented if the programmer wants to inform the user of new application updates. The function which will be invoked when new updates have been downloaded is the "public void UpdateReady(String update)", which describes which modules that have been updated.

6. **Make a reference to the "RequestManager"**
   The last thing to notice about the code at Listing 8.1 is the 9th line, which makes a reference to the "RequestManager" thereby providing the functionality to request data. This reference call has to be requested through an engine call since it is only the engine which has a reference to all plugins loaded in the system. An example of a data request in this project could be:"manager.getData("Intro-image.png")".

8.1.2 Photoalbum

The Photoalbum application is described in the use case chapter. We implement a digital photoalbum using FoDa as the base of the application. Figure 8.1 on the following page displays a screenshot of the application in action.

The photoalbum consists of two buttons: Previous and next. These are used to browse through images. In the top of the screen a label displays the name of the image currently being viewed.

Images are located on a remote server and by using the navigation buttons FoDa will download a requested image and the application will display it to the user.

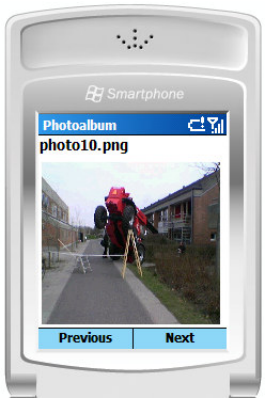


Figure 8.1: Screenshot of the photoalbum application

The Application Code

To illustrate the workload when implementing a Photoalbum with FoDa, this section displays the code which the programmer needs to make a photoalbum. This section omits the GUI code, to setup buttons and the theme since this is normally written by the IDE which uses drag'n'drop components. To implement the photoalbum we follow steps 1, 2, 5 and 6 from the tutorial described in section 8.1.1 on page 84. Since the photoalbum only need to request images and no other kinds of data it is not necessary to implement points 3 and 4 from the tutorial.

Photos displayed in the photoalbum have to comply with the naming convention which the photoalbum uses. Thus, the photos have to be called "photoX.png" where X ("imageNumber" in the code) is an integer value. This is described using three global variables in the photoalbum, as displayed in Listing 8.2.

```
1  private const String imageName = "photo";
2  private int imageNumber = 0;
3  private const String imageExtension = ".png";
```

Listing 8.2: Variables used for the naming convention

The main entry point for the photoalbum plugin is called Run which is executed at start-up.

```
1 private RequestManager.IRequestManager manager;
2
3 public void run(IEngine.IEngine engine)
4 {
5   this.manager = (RequestManager.IRequestManager)engine.GetPlugin("
        RequestManager.dll");
6   lblName.Text = imageName + imageNumber + imageExtension;
7   manager.GetData(imageName + imageNumber + imageExtension);
8   Application.Run(this);
9 }
```

Listing 8.3: The run() function of the photoalbum

The first line in the run method establish a reference to the RequestManager which is the interface to FoDa. The second line simply changes the name of the label to the first image to be displayed. The third line uses the request manager to request the first image from the remote server. The last code line initializes a thread which starts the windows message loop used by the GUI components and input components of the application.

Upon receiving the requested images from the server and converting it to an image object, FoDa triggers the ImageDownloaded event. Listing 8.4 displays the code that is executed when the image downloaded event occurs.

```
1 public void ImageDownloaded(String fileName, Bitmap data)
2 {
3     if (lblName.Text.CompareTo(fileName) == 0 )
4     {
5       imgView.Location = new Point(0,0);
6       imgView.Image = data;
7     }
8 }
```

Listing 8.4: Implementation of the ImageDownloaded() function

Line one of the ImageDownloaded event is a check to see whether the image currently displayed is the one received. This check is necessary since all downloading of data in FoDa is none blocking. This means that there is no way to guarantee that the events occur in the same order as the images are requested. The advantage of this is that the application is able to download multiple images while browsing one image. Thus, it is necessary to check whether the image being viewed is the one that was embedded with the event.

The code inside the check serves to place the image at the right location and of course sets the image data itself to be displayed (to the user).

The code for the navigation buttons are displayed in Listing 8.5. The code is fairly simple since the image number is increased and the new image is requested. An equivalent code exists for the *previous* button, the only difference from *next* is that the image number is decreased. Therefore we have omitted displaying the code for the *previous* button.

```
1 private void Next_Click(object sender, EventArgs e)
2 {
3   imageNumber++;
4   manager.getImage(imageName + imageNumber + imageExtension);
5 }
```

Listing 8.5: Implementation of the “next” button

To allow the user to pan the image, code for reacting to key input is required. The code is displayed in Listing 8.6. The code is fairly simple, by pressing the left, right, up or down key the image is moved 10 pixels in the direction of the key. If numeric key five is pressed the application will exit. But before it does this a call to FoDa is made to shutdown graciously and saving data to a local storage if needed.

```
1  private void Photoalbum_KeyDown(object sender, KeyEventArgs e)
2  {
3    if(e.KeyCode == Keys.Left)  imgView.Left += 10;
4    if(e.KeyCode == Keys.Right) imgView.Left -= 10;
5    if(e.KeyCode == Keys.Up)    imgView.Top  += 10;
6    if(e.KeyCode == Keys.Down)  imgView.Top  -= 10;
7    if(e.KeyCode == Keys.D5)
8    {
9      manager.Terminate();
10     Application.Exit();
11   }
12 }
```

Listing 8.6: The code used to register which keys the user press

Upon receiving an update to parts of FoDa or the application itself, the UpdateReady event will be triggered. The code to react to an update event is displayed in Listing 8.7. The code simply displays a message box to the user indicating that there is an update ready for the system and that a restart of the application is needed. It is then up to the user to actually restart the application upon which the updates will be installed. The update event will trigger for each update downloaded. To make sure the user is only notified once a boolean is set.

This concludes the code required by the programmer to create a Photoalbum that uses a remote server to receive images.

Conclusion

The code that has to be written (that is related to FoDa) is the sum of the code lines displayed in the listings above which totals to 50 code lines. The programmer does of course have to write some GUI code as well, but most of this is auto generated by the IDE such as Microsoft Visual Studio by dragging control elements.

```
1  private bool notificationShown = false;
2
3  public void UpdateReady(String update)
4  {
5    if (notificationShown == false)
6    {
7      notificationShown = true;
8      DialogResult ret = MessageBox.Show("An update is ready. The
9      application have to be restartet for the update to take effect",
10     MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
           MessageBoxDefaultButton.Button1);
11   }
12 }
```

Listing 8.7: When updates is ready to be used the "UpdateReady()" function in the Photoalbum plugin is executed

One might think that 50 lines for writing a simple application like a photoalbum is a lot. But remember the extra features this application now contains as a direct result of using FoDa. The complete feature list of the photoalbum application is displayed below:

1. Browse images from remote server storage.
2. Pan images to view images larger than the screen.
3. On demand delivery of data
4. Automatic conversion of data to useable objects
5. A data cache to lower bandwidth usage and optimize performance of applications
6. Automatic content and patch updates
7. Automatic detection of available network protocols
8. Automatic selection of the best network protocol
9. Automatic manage all network connections
10. Encryption to prevent tampering of content and application

8.1.3 Jump Game

This section describes the implementation of a simple one button jump game using the FoDa framework. The game consists of a set of obstacles and a jumping man. The purpose of the game is to jump over as many obstacles as possible. Figure 8.2 displays a screenshot of the game in action.

Figure 8.2: Screenshot of the jump game

The Application Code

To implement this game we follow steps: 1,2 and 6 from the tutorial described in section 8.1.1 on page 84. Since this application will not request sound or data objects we do not use steps 3 and 4. We also omit using step 5 which means that the user will not be informed about new updates to the code, but will first notice the change the next time the application is started.

The first snippet of code we show is the run method as shown in Listing 8.8. As described in Section 8.1.1 on page 84 this method is the main entry point for the application and it is responsible for setting up the connection to the engine and the interface to the framework using the RequestManager. The method also makes the call that starts the windows message processing for the application.

```
1  private IEngine.IEngine engine;
2  private RequestManager.IRequestManager manager;
3
4  public void run(IEngine.IEngine engine)
5  {
6    this.engine = engine;
7    this.manager = (RequestManager.IRequestManager)engine.GetPlugin("
         RequestManager.dll");
8
9    Application.Run(this);
10 }
```

Listing 8.8: The run method

The Jumper_Load method at Listings 8.9 is called by the windows message processing loop when the application is loaded. Here we use the framework to load images used in the game and start the game processing loop which is responsible for executing the game logic and painting the screen.

The last method related to our framework in the game code is the Image-Downloaded event method seen in Listing 8.10 on the next page. This method

```
1  private void Jumper_Load(object sender, EventArgs e)
2  {
3    WindowState = FormWindowState.Maximized; // Use fullscreen mode.
4
5    bmpBackBuffer = new Bitmap(this.ClientRectangle.Width, this.
         ClientRectangle.Height);
6    gxBackBuffer = Graphics.FromImage(bmpBackBuffer);
7
8    manager.getImage("background.png");
9    manager.getImage("cloud1.png");
10   manager.getImage("cloud2.png");
11   manager.getImage("jumper.png");
12   manager.getImage("crate.png");
13
14   // Start Game Loop
15   DoGameLoop();
16 }
```

Listing 8.9: The jumper load method.

is responsible for initializing images as they arrive. The images are initialized with the image data received as well as their stating position which is outside the screen area on the right.

```
1  public void ImageDownloaded(String fileName, Bitmap data)
2  {
3    if (fileName.Equals("background.png")) {
4      imgBackground = data;
5    }
6    else if (fileName.Equals("cloud1.png")) {
7      clouds[0] = new Cloud(data, ClientRectangle.Width, 0,
           transparentColor);
8    }
9    else if (fileName.Equals("cloud2.png")) {
10     clouds[1] = new Cloud(data, ClientRectangle.Width + data.Width + 42,
           data.Height, transparentColor);
11   }
12   else if (fileName.Equals("jumper.png")) {
13     player = new Player(data, ClientRectangle.Width / 2, ClientRectangle
           .Height - 38 - data.Height, transparentColor);
14   }
15   else if (fileName.Equals("crate.png")) {
16     for (int i = 0; i < NumberOfCrates; i++)
17     {
18       crates[i] = new Crate(data, ClientRectangle.Width + (2* i * data.
             Width), ClientRectangle.Height / 2, transparentColor);
19     }
20   }
21 }
```

Listing 8.10: Image downloaded event method

This concludes the lines of code that are directly related to the use of FoDa.

Conclusion

The code for the jump game consists of 554 numbers of lines in total. 47 of the lines relate to the use of FoDa and the code described in the previous section is made by these 47 lines. The rest of the code is pure game related such as game logic, input handling, collision detection etc.

1. Simple one button jumping game
2. On demand delivery of data
3. Automatic conversion of data to useable objects
4. A data cache to lower bandwidth usage and optimize performance of applications
5. Automatic content and patch updates
6. Automatic detection of available network protocols
7. Automatic selection of the best network protocol
8. Automatic manage all network connections
9. Encryption to prevent tampering of content and application

8.1.4 Indoor Maps

The next implementation we describe is that of a simple Map application. Figure 8.3 illustrates how this application looks like. The purpose with this application is to navigate a user through a building which is unknown by the user. The idea of this application comes from Rene Hansen, a PhD student, who at the moment is researching in how to navigate by use of Wi-Fi spots instead of GPS. The reason for this is that GPS is not accurate enough to work indoor. This is described more in the use cases, Section 2.2 on page 11. Since Rene Hansen was not far enough in the research process, it was not possible to use his Wi-Fi measurements and transmit it to a database for translating the Wi-Fi measurement into map data. So, instead of using actual Wi-Fi data we simulate a person moving around in the computer scientist department of Aalborg University. The simulation thread used for this application can be seen in Appendix A The application basically works by “measuring” the location of the user, then transmits the data to a server which then provides the map data needed for the user’s current location. When the user moves too far away from the last position he received data for, the system asks for new map data for the user’s new location.

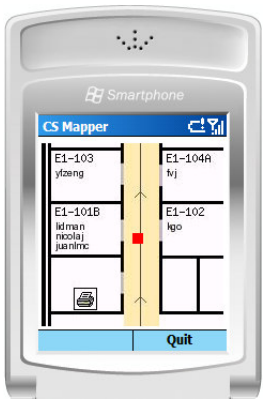


Figure 8.3: Screenshot of the photoalbum application

The Application Code

The code for this application only consists of two functions and one thread. For the same reason as described concerning the Jump Game we only make use of steps 1, 2 and 6 from the tutorial in this application. However, to optimize the performance of this application a new Prefetch plugin have to be made. Furthermore, a few changes to the cache will also speed up the performance of the application.

Besides all the GUI setup only the following codes in Listing 8.11, 8.12 and 8.13 are used to implement the Indoor Map application.

```
1  private IEngine.IEngine engine;
2  private RequestManager.IRequestManager manager;
3
4  public void run(IEngine.IEngine engine)
5  {
6    this.engine = engine;
7    this.manager = (RequestManager.IRequestManager)engine.GetPlugin("
         RequestManager.dll");
8    manager.getImage("dot.png");
9
10   DT = new DataThread();
11   Thread DataT = new Thread(new ThreadStart(DT.run));
12   DataT.Start();
13
14   Request request = new Request(manager, DT);
15   Thread getlocation = new Thread(new ThreadStart(request.StartRequest))
         ;
16   getlocation.Start();
17
18   Application.Run(this);
19 }
```

Listing 8.11: The run method

Line 8 in Listing 8.11 request the image “dot.png” which is a red dot used to illustrate where the user is located. Lines 10 to 12 are used to start the simulation

thread which simulates the user's location. In a real map application this should not be here. Lines 14-16 start a new thread which updates the user's location based on the Wi-Fi measurement (in this example the simulated data).

```
1  public void ImageDownloaded(String fileName, Bitmap data)
2  {
3    if (fileName == "dot.png")
4    {
5      pictureBox2.Image = data;
6    }
7    else
8    {
9      Vector Location = new Vector(fileName);
10     Vector CurrentLoc = DT.getXY();
11
12     pictureBox1.Left += Location.X - CurrentLoc.X;
13     pictureBox1.Top += Location.Y - CurrentLoc.Y;
14     pictureBox1.Image = data;
15   }
16 }
```

Listing 8.12: Image downloaded event method

In the ImageDownloaded method in Listing 8.12 it is necessary to check whether the image downloaded is the "dot.png" or map data. Since the user might have changed location at the point in time when the new map data is received, it is necessary to calculate how much the user has moved, in order to display the map data correct in relation to the user's current location. This is done in lines 9-13 in Listing 8.12.

The last code needed in the Application plugin is a thread which asks for data. The code for this thread can be seen in Listing 8.13. The only purpose of this thread is to request data every $\frac{1}{4}$ second.

Optimizations

If the application described in the previous section is used with FoDa as described in Chapter 7 on page 58 it will run very slowly because the application requests new images every $\frac{1}{4}$ second. When "getData()" is used, FoDa will check in the cache whether the data for the user's specific location is already stored. Most of these checks will return false since the user has probably moved to a new location when the data has been downloaded and is ready to be used. Thus, to optimize the performance it is necessary to change the Cache plugin in order to make the cache return the data which is closest to the user's location instead of the data at the user's exact location.

By using this performance optimization trick the Cache plugin thinks that it has the data needed, and therefore just returns the data closest to the user's location, even if it is too far away from the user to be relevant. So to be sure the cache receives new data it is necessary to make a new Prefetch plugin. Every

```
1  public class Request
2  {
3    RequestManager.IRequestManager manager;
4    DataThread DT;
5
6    public Request(RequestManager.IRequestManager manager, DataThread DT)
7    {
8      this.manager = manager;
9      this.DT = DT;
10   }
11
12   public void StartRequest()
13   {
14     Vector location;
15     while (true)
16     {
17       location = DT.getXY();
18       manager.getData(location.ToString());
19       System.Threading.Thread.Sleep(250);
20     }
21   }
22 }
```

Listing 8.13: Image downloaded event method.

time the application requests new data this is transmitted to the Prefetch plugin as well. When the Prefetch receives the first request from the application it has to request the data from the Internet. Every time the Prefetch plugin receives a request for data it checks whether the location of the user is too far away from the last internet request made by the Prefetch plugin. If the user has moved too far away from the last internet request, the Prefetch knows that the user will soon need new data and therefore makes a new Internet request for data.

Conclusion

As described only two functions and one thread have to be made for this application to work, though one new Prefetch plugin and some modifications to the Cache plugin are needed to gain optimal performance. The feature the Indoor Map application uses from FoDa is displayed below:

1. Modular design for adapting to various mobile devices and easy extending and modification of the framework
2. On demand delivery of data
3. Automatic conversion of data to useable objects
4. Prefetching of application content
5. A data cache to lower bandwidth usage and optimize performance of applications

6. Automatic content and patch updates
7. Automatic detection of available network protocols
8. Automatic selection of the best network protocol
9. Automatic manage all network connections
10. Encryption to prevent tampering of content and application

8.1.5 Porting a Game: Pocket 1945

The three latter sections have described how we have implemented three different applications by use of FoDa. As described in the introduction of this chapter one might find the three applications too simple to give a general impression of the usefulness of FoDa. Because of this fact this section describes how we have converted a standalone application to use our system, and thereby provide the application with all the features of FoDa. One might argue that this application is simple as well, but we think it very well represents the type of games one see on mobile devices in the sense of gameplay and simplicity.

The Application we convert is the game Pocket 1945 written by a Norwegian citizen Jonas Follesø. The game is available with full source codes from the Code Project [34]. The game is written for Pocket PCs using the .Net Compact Framework.

The game is a flight shooter set around 1945. A screenshot from the game is shown in Figure 8.4.

Figure 8.4: Screenshot from Pocket 1945

The Application Code

Since this application only uses images we will make use of only steps 1,2 and 6 from the tutorial described in Section 8.1.1 on page 84.

Porting of the game starts by including the references to the engine and the RequestManager. Then the application entry point is changed from the main method to the run method of the plugin interface. Basically only two classes of the game is modified, the GameForm which contains game constructor and entry point as well as the class SpriteList which handle the image resources.

We start by looking at the changes to the GameForm class. Listings 8.14, 8.15 and 8.16 display modifications to this class. The new signature for the GameForm class is shown in Listing 8.14.

```
1 public class GameForm : Form, IEngine.IPlugin, RequestManager.
      IImageDownloaded
```

Listing 8.14: Modification to the *Main* method

We no longer require the Main method as the entry point. The Plugin run method will take its place.

```
1  public static void Main() {
2    //Application.Run(new GameForm());
3  }
4
5  private IEngine.IEngine engine;
6  private RequestManager.IRequestManager manager;
7
8  public void run(IEngine.IEngine engine)
9  {
10   this.engine = engine;
11   this.manager = (RequestManager.IRequestManager)engine.GetPlugin("
         RequestManager.dll");
12
13   SpriteList.Instance.LoadSprites(manager);
14
15   levelFiles = GetLevels();
16
17   DoGameLoop();
18
19   Application.Run(this);
20 }
```

Listing 8.15: Modification to the *Main* method and the plugin *run* method

Note the change in the Application.Run statement to use *this* instead of *new*. This is due to the fact that the engine will instantiate the plugin and then call the run method. Another change which is needed is the loading of contents from loading in the constructor to loading in the run method. The game loop must also be started from the run method.

```
1 public void ImageDownloaded(string fileName, Bitmap image)
2 {
3   SpriteList.Instance.ImageDownloaded(fileName, image);
4 }
```

Listing 8.16: Implementation of the *ImageDownloaded* event

The ImageDownloaded event is forwarded to the SpriteList class which handles all images. The SpriteList is a singleton pattern holding a list of all the images for the game. The SpriteList modifications are displayed in Listings 8.17 and 8.18.

```
1  public void LoadSprites(RequestManager.IRequestManager manager)
2  {
3    if(!doneLoading)
4    {
5      manager.getImage("Tiles.bmp");
6      manager.getImage("Bonuses.bmp");
7      manager.getImage("Bullets.bmp");
8      manager.getImage("SmallPlanes.bmp");
9      manager.getImage("SmallExplotion.bmp");
10     manager.getImage("BigBackgroundElements.bmp");
11     manager.getImage("BigExplotion.bmp");
12     manager.getImage("BigPlanes.bmp");
13   }
14 }
```

Listing 8.17: Modifications to the LoadSprites method.

The *LoadSprites* method calls the RequestManager with image requests. This will cause a ImageDownloaded events to trigger when the images are downloaded.

Conclusion

Comparing the original game with the new and modified version using Visual Studio Line Counter add-in [35], we note that the original game consists of 2065 code lines whereas the new modified version consists of 2105 code lines. This means that the developer needs only to add 40 code lines to get the full feature set of our system as described in Section 2.5 on page 20. Most of the code modifications are present in the ImageDownloaded method.

It might have been a stroke of luck that the ported game is so well written that all contents is handled in one specific class making it fairly easy to convert. If the use of images had been spread out in the code it would have been far more difficult to convert due to the fact that all image contents has to come through the ImageDownloaded event. Basically, we are converting an application not designed from the scratch to use events. The obvious solution is to make an image class container similar to the one used in the game and replace all uses of images to go through this class. This would require us to make one more class

```
1  private int NumberOfImages = 0;
2
3  public void ImageDownloaded(string fileName, Bitmap image)
4  {
5    if (fileName.Equals("Tiles.bmp")) {
6      Tiles = ParseSpriteStrip(image);
7    }
8    else if (fileName.Equals("Bonuses.bmp")) {
9      Bonuses = ParseSpriteStrip(image);
10   }
11   else if (fileName.Equals("Bullets.bmp")) {
12     Bullets = ParseSpriteStrip(image);
13   }
14   else if (fileName.Equals("SmallPlanes.bmp")) {
15     SmallPlanes = ParseSpriteStrip(image);
16   }
17   else if (fileName.Equals("SmallExplotion.bmp")) {
18     SmallExplotion = ParseSpriteStrip(image);
19   }
20   else if (fileName.Equals("BigBackgroundElements.bmp")) {
21     BigBackgroundElements = ParseSpriteStrip(image);
22   }
23   else if (fileName.Equals("BigExplotion.bmp")) {
24     BigExplotion = ParseSpriteStrip(image);
25   }
26   else if (fileName.Equals("BigPlanes.bmp")) {
27     BigPlanes = ParseSpriteStrip(image);
28   }
29
30   NumberOfImages++;
31
32   if (NumberOfImages == 8) {
33     doneLoading = true;
34   }
35 }
```

Listing 8.18: The ImageDownloaded event handler in the SpriteList class.

and the modifications to the image usage code need to be spread out in the game. Most of these modifications could probably be made using refactoring.

8.1.6 Massive Multiplayer Online Game

As described in the introduction of Chapter 8 we did not have time to implement a massive multiplayer online game by use of FoDa, instead we implemented a single player game as described in Section 8.1.3 on page 89. The single player game is in many ways similar to how a massive multiplayer online game should be implemented by use of FoDa, only in a smaller scale. In this section we describe what kind of extra work is needed to turn the Jump Game into a massive multiplayer online game.

The Application Code

Since we have no implementation code for a massive multiplayer online game, we will not show any code in this section, but will describe how the code should work.

First of all, the developer of the game needs to follow each step as described in Section 8.1.1 on page 84. Following steps 1-6 will give the developer the possibility to download both images sound and game relevant data.

When comparing the Jump game with a massive multiplayer online game one notices the size of the virtual world. If the Jump game should work as a massive multiplayer online game, the virtual world in which the avatar is moving should be much bigger to make space for a number of avatars. A virtual world of this size would probably be too much for a mobile device to handle, but by splitting the world up in small zones it would be possible for each mobile device only to store a small part of the world at a time. Another approach, which we already have implemented and described in a previous section, would be to send out world data in relation to the avatars' current position in the virtual world. This could be implemented in the same way as we have implemented the Indoor Map application as described in Section 8.1.4 on page 92.

Another change which is needed to make the Jump game work as a massive multiplayer online game is the communication with other clients to receive their location in the virtual world. Since this communication is not in the form of a picture or image the developer has to use the method described in the *IDataDownloaded* interface. To receive the locations of other avatars each client would have to request the location of nearby avatars frequently, a request which therefore should be part of the game loop. Listing 8.19 on the next page shows some pseudo code for how this request could be made and how the data received should be handled.

The pseudo code in Listing 8.19 on the facing page asks for other players' location every second. This means that if another player gets close to the client the server will send the location of the other player to the client. When the client receives a *DataDownloaded* event the client first asks for an image of the other player, after which other relations to that player can be set. When the *ImageDownloaded* event receives the image of the other player close to the avatar it simply displays the image for the other player at the correct location.

FoDa is made with the purpose to handle context and updates it will be a little tricky to handle all kinds of communication with other clients in the current implementation of FoDa. For instance, if the client wants to be able to communicate with other players in the game the *getData("User message")* should be used. Furthermore to receive messages from other players the client has to check for messages from time to time. A message will be received by the *DataDownloaded* method and will then be displayed to the user.

```
1  public void GameLoop()
2  {
3    while(true)
4    {
5      //Request data for players nearby to the client
6      manager.getData(PlayerLocation.X,PlayerLocation.Y);
7      //wait 1 second till the next request
8      sleep(1000);
9    }
10 }
11
12 //The response to the request is received by the DataDownloaded
13 //as it is not a image or a sound
14  public void DataDownloaded(String fileName, byte[] data)
15 {
16   //The developer have to convert this data himself
17   List PlayerLocationNearBy = ByteToList(data);
18   for(int i=0; i<PlayerLocationNearBy.size;i++)
19   {
20     //Request the image of player ``i''
21     //And tag the data as a location of another player
22     getData("PlayerLocation:" + PlayerLocationNearBy[i].Location);
23
24     //If other relation should be set it should be done here
25     if(PlayerLocationNearBy.group != Avatar.group)
26     {
27       //Warn the player of a nearby enemy
28     }
29   }
30 }
31
32 public void ImageDownloaded(String fileName, Bitmap data)
33 {
34   if(fileName.Contain("PlayerLocation:")
35   {
36     Location[] = fileName.split(':');
37     Display the Bitmap image at Location[1];
38   }
39 }
```

Listing 8.19: Pseudo code for requesting and handling of player locations

Conclusion

By using FoDa aspects related to retrieving contents for a massive multiplayer online game should be as simple as with the Jump game as described in Section 8.1.3 on page 89. However, if the client has to send data to other clients the implementation of FoDa as described in Chapter 7 would make it possible though somewhat tricky because the client has to use the method *getData()* to actually send data to other clients.

8.2 Benchmark

The Benchmark section focuses on the performance of FoDa. This section is intended to give application programmers an insight into how much resources

and overhead FoDa introduce to an application.

In most cases benchmark results will be compared to a version of the application not using FoDa. An example is that we have produced a copy of the Photoalbum which do not use FoDa, but still keeping the code as close to the original as possible. This version will be compared to the one using FoDa. We also test the version of Pocket 1945 with and without FoDa.

Throughout this chapter we use FoDa in front of applications to refer to those using FoDa. An example would be the Photoalbum using FoDa is refereed to as: FoDa Photoalbum.

8.2.1 Memory Usage

To see how much memory FoDa use during execution we have executed the Photoalbum and compared it to FoDa Photoalbum and we executed the Pocket 1945 and compared it to the FoDa Pocket 1945 version. To generate the memory statistics we have used .Net CFs built-in ability [36]. The statistics produced by running the tests can be seen in Table 8.1 and Table 8.2.

	Photoalbum	FoDa Photoalbum	Difference
Peak memory used	275.6KB	562KB	286.4KB
Memory used	99.7KB	271.6KB	171.9KB
Object allocated	1,486	3,662	2,176

Table 8.1: Memory statistics from the Photoalbum and the FoDa Photoalbum applications

	Pocket 1945	FoDa Pocket 1945	Difference
Peak memory used	776.2KB	1,151.6KB	375.4 KB
Memory used	499.2KB	1,044.75KB	545 KB
Object allocated	6,504	13,903	7,399

Table 8.2: Memory statistics from the Pocket 1945 and the FoDa Pocket 1945 games

Table 8.1 shows that FoDa Photoalbum uses more memory than the Photoalbum application. FoDa Photoalbum increases the memory use with 171.9KB while running and with 286.4KB at peak memory. Some of the increased memory use is due to fact that FoDa Photoalbum is loaded through plugins. In addition Table 8.1 shows that the device needs to manage 2,176 more objects, which will slow down the overall performance of the Photoalbum.

Table 8.2 shows that the Peak memory used is higher with the FoDa Pocket 1945 application compared to the FoDa Photoalbum application. The reason for this is that the FoDa Pocket 1945 requests 8 pictures during the start-up, whereas the FoDa Photoalbum requests only 1 image. This results in more memory used, both comparing Peak memory and when the application is just running. Like in

the Photoalbum the use of FoDa also results in more objects which need to be managed.

If we compare the Pocket 1945 and Photoalbum application, we see that the memory use increases with the complexity of the application using FoDa. Furthermore we find that the device needs to manage more objects when FoDa is used which will slow down the system. This is however to be expected since FoDa introduces a number of extra features compared to the application not using FoDa. These are features like Cache, Security manager and ability to add and remove plugins.

8.2.2 System Size

In this section we describe how much extra disk space the use of FoDa needs. To measure the disk space used we compare the Photoalbum with the FoDa Photoalbum as well as the Pocket 1945 with the FoDa Pocket 1945 game.

	Photoalbum	FoDa Photoalbum	Difference
Application size	12.8KB	124.7KB	111.9KB

Table 8.3: Disk space statistics for applications of the Photoalbum

Table 8.3 shows that FoDa roughly increases the disk space with 112KB which might seem a lot for a simple application as the Photoalbum is. However, since FoDa is modular by design, the application developer should tailor FoDa to his own needs if he finds the 112KB to be too much for his application. Looking at a less simple application Table 8.4 displays statistics of the two Pocket 1945 games, both with and without FoDa.

	Pocket 1945	FoDa Pocket 1945	Difference
Application size	360 KB	157.5KB	202.5KB

Table 8.4: Disk space statistics for the Pocket 1945's games

As Table 8.4 shows the size of the application has been reduced by use of FoDa. The reason for this is that the Pocket 1945 application uses graphic which have been bundled up with the application, thereby increasing the application size. By using FoDa it is possible to download the data when it is needed.

8.2.3 Startup Time

To measure how much longer the start-up time is by using FoDa we have implemented a timer into the Photoalbum and FoDa Photoalbum applications. The first thing the two applications do is to start a timer, which stopped when the applications are done painting their graphics on the screen.

	Photoalbum	FoDa Photoalbum	Difference
Start-up Time	417ms	5,799ms	5,382ms

Table 8.5: Start-up time

Table 8.5 shows that FoDa Photoalbum takes more than five seconds more to start than the Photoalbum application do. This is because almost all of the components of the FoDa Photoalbum are loaded at start-up time, and all of them have their own dll files, which might increase the load time.

To test whether the long start-up time is caused by the division of the system into many plugin, which all have their own dll files, we have collected all plugins in one dll file, the engine in another and the application plugin in a third dll file. This should reduce the start-up time, as only a few plugins has to be loaded. The collection of plugins in one dll file has removed the possibility to completely remove a plugin. It is only possible to tell the engine not to load it or load another one instead, but the plugin will still be in the combined dll file, and therefore still contributing to the overall size of the system.

	Combined Photoalbum	FoDa Photoalbum	Difference
Start-up Time	5,113ms	5,799ms	686ms

Table 8.6: Start-up time for combined plugins

Figure 8.6 shows that the start-up time has been reduced with 686ms, but the start-up time still exceeds 5.1 seconds, which is 4.7 seconds slower than the application without the use of FoDa (see Figure 8.5 start-up time without FoDa). Therefore, we can conclude that collecting all plugins in one plugin does not perform much better during start-up. Furthermore, the collection of dlls removes the possibility to completely remove plugins from the system; we do not recommend to combine plugins in one dll file.

Start-up Time Details

To give a more detailed picture of why our system use over 5 seconds to start-up we will in this section time trace the start-up of FoDa. To time trace the start-up of FoDa we have implemented a log feature in FoDa and used the FoDa Photoalbum as a test application. Tabel 8.7 on the next page shows a small part of the time trace log. The full log can be seen in Appendix B. The log shows that FoDa use about 1.78 seconds right at the start. Looking at the code from entrance two and entrance three in Table 8.7 on the facing page we have found that the large time gab comes between two simple instructions.

The code from this part of the time trace can be seen at Listings 8.20

We have at this point in the code not started any threads, so we assume that the time gab comes from outside of FoDa and must be a thread started by the

Time in ms	Log
10	Engine start
141	Check for updates (Starting)
1921	UpdateDLLs (Starting)
1967	UpdateDLLs (Done)
1967	Check for updates (Done)
1969	initialize plugins (Starting)
1987	ParsePluginXML (Starting)
3212	ParsePluginXML (Done)
3231	Load requires (Starting)
3250	LoadPlugin:RequestManager.dll (Starting)
4046	LoadPlugin:RequestManager.dll (Done)
4046	Load requires (Done)
4047	Load core
4049 - 4078	Load the Datamanager and RequestManager
4078	LoadPlugin:Photoalbum.exe (Starting)
5272	LoadPlugin:Photoalbum.exe (Done)
5273	Load core (Done)
5273	initialize plugins (Done)
5274	Engine done, starting application
5279	RunCore
5290 - 5691	Load the UpdatePolici, Cache, SecurityManager, ConnectionManager, Http and the Prefetch plugin
5742	Show GUI
7413	LoadPlugin:Encryption.dll (Starting)
7472	LoadPlugin:Encryption.dll (Done)

Table 8.7: A segment of the time trace log of FoDa Photoalbum start-up time. All of the time trace be seen at Appendix B

.net CF, probably because of some earlier instructions in FoDa, it have however not been possible for us to find the instruction which spawn the thread. The next large time gab in the time trace is the parsing of XML which roughly takes 1.2 seconds; this part will vary depending on how many plugin is used. This will be described in the next section.

The first plugin is then loaded through loading of requirements, this plugin takes about 800ms, where the rest of the core plugins takes about 20ms to load. The long load time of the first plugin is proberly due to initializing of the mechanics used for loading plugins. Section 8.2.3 on page 107 describe more thoroughly how much extra time the introduction of extra core plugin takes to load. As seen in the time trace log the Photoalbum takes roughly 1.2 seconds to load, most of this time is probably due to initializing of all the GUI elements used in the

```
1  public void initialize()
2  {
3    engine.Log("Check for updates (Starting)");
4    UpdateDLLs(XML_UPDATE); // Check if there are files to update.
5    engine.Log("Check for updates (Done)");
6
7    engine.Log("initialize plugins(Starting)");
8    ParsePluginXML(xmlfile);
9    engine.Log("initialize plugins(Done)");
10 }
11
12 private void UpdateDLLs(String xmlfile)
13 {
14   engine.Log("UpdateDLLs (Starting)");
15   if(System.IO.File.Exists(XML_PATH + xmlfile))
16   {
17     Code for Updating of code
18   }
19   engine.Log("UpdateDLLs (Done)");
20 }
```

Listing 8.20: The run method.

application. Most of the plugins in FoDa is not core plugins, and is therefore loaded outside of the core section. These plugins is loaded when the application needs them. As seen in Table 8.7 FoDa uses most of the plugins right away as the FoDa Photoalbum request a image right away after it have been started. After roughly 5.7 seconds all plugins have been loaded and the GUI is shown at the mobile device. The reason the Encryption plugin first is loaded 1.6 seconds after the GUI have been shown is because FoDa at this point have retrieved the image the FoDa photoalbum requested and have to encrypt the image before it is stored in the cache.

Additional plugins

Since one purpose of FoDa's is to be able to use it as part of a bigger system, it is relevant to see how FoDa performs when more than only a few plugins are added to the system. To test this we have added a number of plugins to the system. These plugins have only been added to the system, and are never executed (Executing the plugins would involve overhead from the user's code). We therefore only measure the overhead in extra plugins added to FoDa. This has been done by adding plugins to the engine XML file and not having set them to *Core* (see section 6.4.4 on page 52 for the engine XML file).

Figure 8.5 shows the start-up times when we add extra plugins to the system. We have drawn two trend lines, the first $0.00021x^2 + 5.2x + 5382.95$ shows the best fit to our measurements, and it will fit with an even larger number of plugins (10,000 and 15,000). The second trend line $5.2x + 5382.95$, shows a linear development. This line illustrates that the first factor ($0.00021x^2$) is so small that it is not noticeable if the user use less than 800-1000 extra plugins. So, if the user

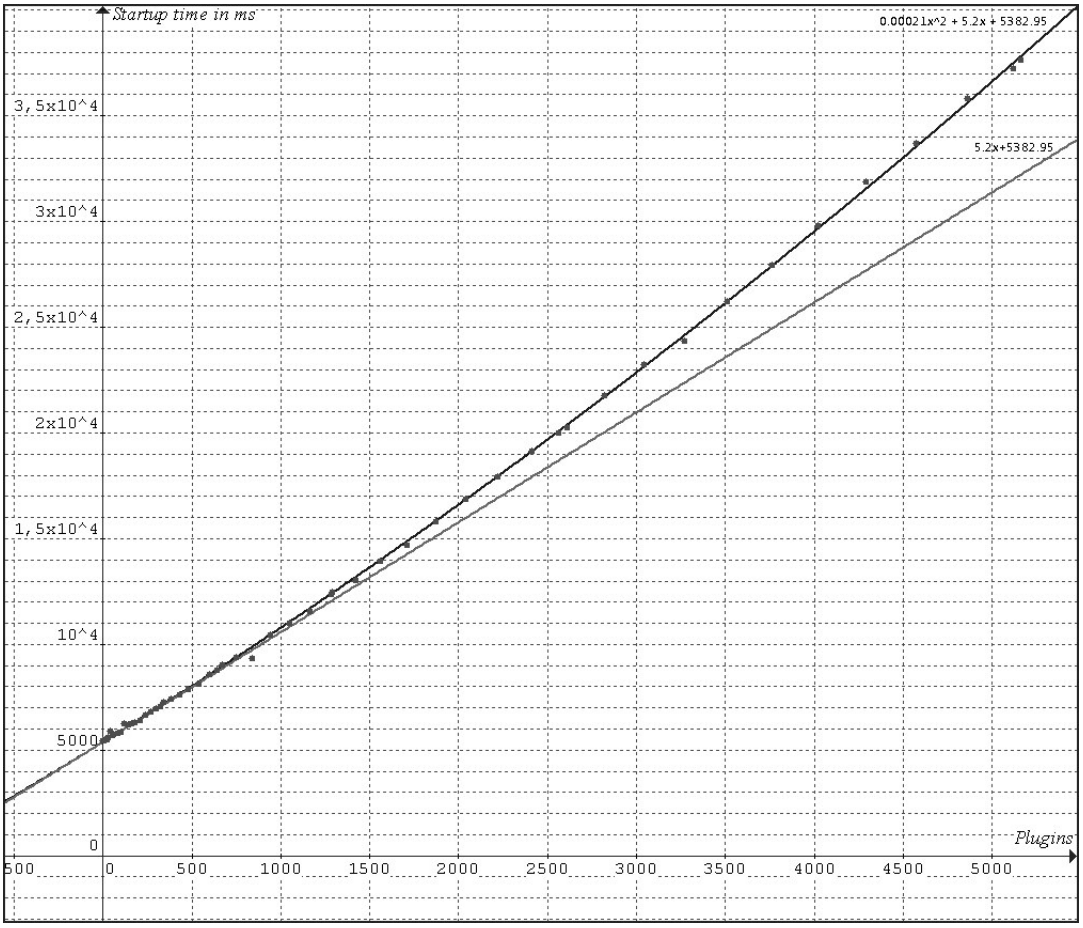


Figure 8.5: Graph over number of extra plugins and start-up time

uses below 800-1000 extra plugins the extra time each plugin introduce to the start-up time is roughly 5.2ms. This extra time is 0.09% of the time to start-up FoDa without extra plugins. Looking at our code for handling plugins there is no construct which should increase the time with a polynomial factor. But, as we do get a polynomial time evolution instead of a linear evolution, our guess is that some of the .Net CF construct we use have a time complexity of $O(n^2)$ which we thereby inherit.

Additional core plugins

In the test described in the previous section, we added more plugins to FoDa, which was not core plugins. the effect of that is that the effect that the plugins were never activated or read into the memory. To test how FoDa performs when the plugins are activated and read into the memory we performed the same test, only now with core plugins.

When a plugin is activated, its run method is executed, and since the developer

is not restricted in adding codes in the run method, it is not possible to test how FoDa will perform when random plugins are added. Instead we have tested how FoDa performs when the smallest plugins possible are added and activated at start-up.

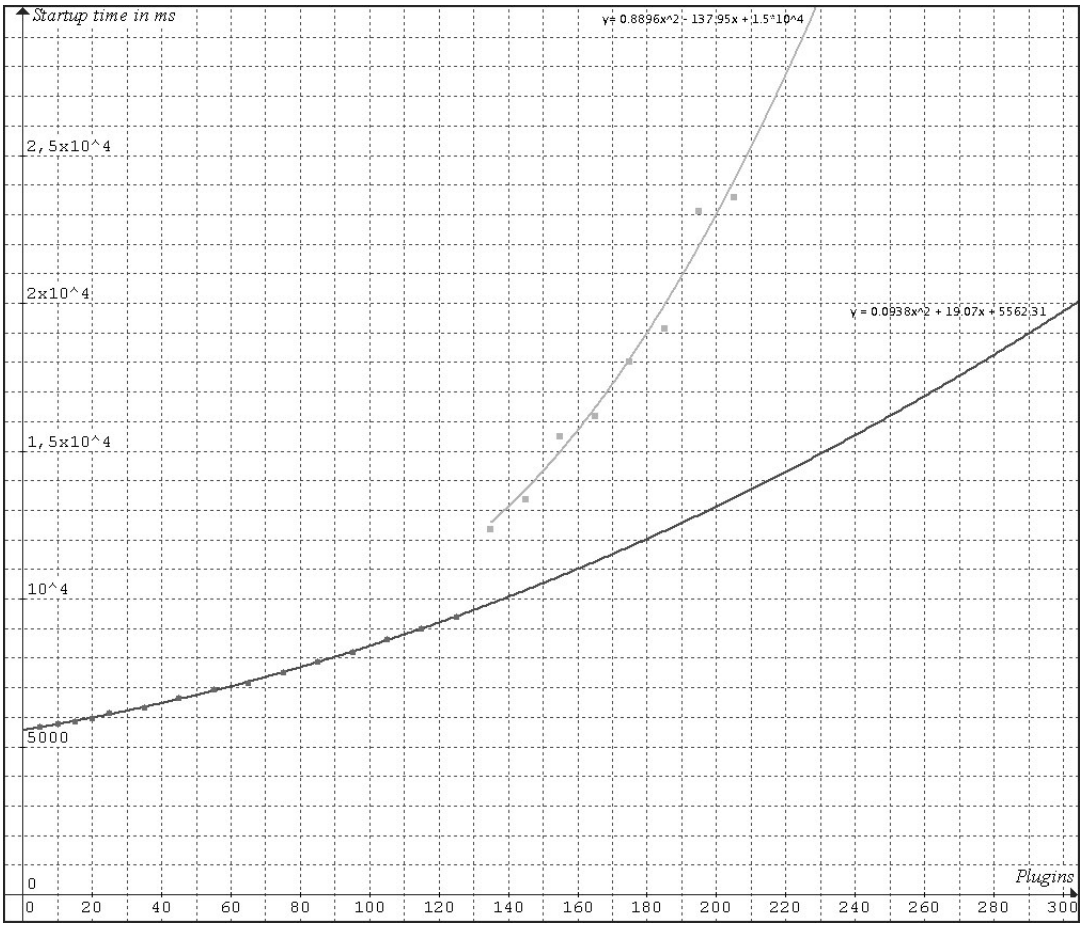


Figure 8.6: The extra startup time for adding new core plugins. Since the Garbage Collector "kicks in" about 130 core plugins we have made 2 trend lines. One trend line for less than 130 core plugins and one for more.

Figure 8.6 shows how much the start-up time increases by adding more core plugins to the system.

The graph shows an extra high start-up time when going from 125 core plugins to 135 core plugins. This big difference is because the Garbage Collector has to start-up simultaneous with the start-up phase. Until the Garbage Collector starts the trend line is $y = 0.0938x^2 + 19.07x + 5562.31$. Roughly 19 ms extra start-up time is introduced per extra core plugin which is used in FoDa (as long as not more than 130 core plugin are used). This is 13.87ms more than when using non core plugins. Figure 8.6 also shows that when the Garbage Collector starts to run the trend line changes to $y = 0.8896x^2 - 137.95x + 1.5 \cdot 10^4$. The effect is

that the start-up time of FoDa is increased by 19ms for each new core plugin.

8.2.4 Execution Time

One of the most important features in FoDa is the ability to download data from the internet. To test how much overhead FoDa introduces to applications using the system, we have tested how much extra time is used for downloading data through our system compared to an application implementing the download of data itself.

Internet performance

To measure how much overhead FoDa introduces to the internet performance of an application we have compared the Photoalbum and the FoDa Photoalbum to measure the delay of downloads.

To measure the download delay we start a timer when a request is made and stop the timer when the picture has been downloaded and is ready to be used. Since the data is provided by use of the internet, a little variation in the time from the request is made to the data is ready have to be expected. To compensate for this and validate the test result we have used a statistical sample size formula to find the amount of tests needed. The formula we have used to calculate the sample size is displayed in Equation 8.1. Equation 8.1 should be used if the population is large or unknown. The population size is only likely to be a factor when working with a relatively small and known population. If this is the case Equation 8.2 should be used.

$$\text{Sample Size} = \frac{\text{Confidence Level}^2 \cdot \text{Percentage} \cdot (1 - \text{Percentage})}{\text{Confidence Interval}} \tag{8.1}$$

$$\text{Sample Size(Population)} = \frac{\text{Sample Size}}{1 + \frac{\text{Sample Size}-1}{\text{Population}}} \tag{8.2}$$

Confidence Level: Determines how confident you need to be that the results are representative.

Confidence Interval: Describes the amount of error which can be tolerated.

Percentage: Describes how large the percentage of a particular answer is.

Population: Describes the population size.

Since it is always possible to retrieve extra data for this test the population size should be considered large, for this reason we have used Equation 8.1. We have chosen to use a *confidence level* of 95% which are used in modern applied practice [37] and a *Confidence Interval* of 5% which we find as reasonable margin

of error. This means that we are within 5% of the true answer in 95% of the tests we run. Using these data gives us a Sample Size of 384. Table 8.8 display the average of 384 test of downloading image 1 to 10.

Image	Photoalbum	FoDa Photoalbum	Difference
1 - 919B	190.3ms	178.3ms	6.7%
2 - 3.2KB	342.7ms	337.9ms	1.4%
3 - 6.3KB	295.2ms	306.5ms	3.6%
4 - 4.3KB	1009.3ms	1009.1ms	0%
5 - 144KB	1239.3ms	1068.1ms	6.6%
6 - 18.4KB	280.7ms	262.1ms	7%
7 - 58.9KB	630.4ms	614.1ms	2.6%
8 - 237.7KB	1474.0ms	1351.7ms	9%
9 - 99.1KB	792.6ms	826.2ms	4%
10 - 153.4KB	1272.1ms	1231.4ms	3.2%
Average	**752.7ms**	**718.5ms**	**4.7%**

Table 8.8: Internet communication statistics, each value in the table is the average value of 384 tests

Table 8.8 shows that there is only a 4.7% difference between the overall average of using FoDa and not using FoDa, which is lower than the margin of error. From this we can conclude that there is practically no overhead on the Internet communication using FoDa compared to a similar application not using FoDa.

8.3 Comparing the Eclipse eRCP

The last sections described the performance of FoDa. This section we will describe how well the Eclipse eRCP framework covers the features needed to handle the use cases of Section 2 on page 9 and compare FoDa with the features of the eRCP.

8.3.1 The Eclipse eRCP

Eclipse is a pure plugin system for desktop computers [19]. The most known project build on top of the Eclipse platform, is a IDE for developing Java applications by the same name.

To support other language than Java different plugins have been developed, such as plugins for C++, Cobol, AspectJ and many more. Eclipse has also been used in the development in several other projects, such as BIRT (Business Intelligence and Reporting Tools), DTP (Data Tools Platform), EMF (Eclipse Modelling Framework), and many more.

The most interesting project in mobile device context is the eRCP (Embedded Rich Client Platform), which attempts to move the Eclipse plugin system to mobile devices. eRCP have now arrived in version 1.0.2, and includes:

Core Runtime Eclipse The core which provides OSGI [38] and extension point support

eSWT The Standard Widget Toolkit which is a subset of the desktop version, and enable the use of normal GUI elements such as Labels, Tables, Tree, Dialogs, and Web browser. It also contains a set of device specific elements.

eWorkbench A UI framework supporting multiple eRCP applications at the same time, and start-up of new eRCP applications.

eJFace A set of classes which extend eSWT with more complex widgets, and enables eRCP applications to integrate with eWorkbench.

eUpdate An interface for dynamic updating of software.

microXML An XML interface supporting SAX and DOM parsing.

To support difference devices and more complex capabilities of new devices, eRCP is divided into two different profiles, a *Core Profile* which contain: Core Runtime, Core eSWT, SWT mobile Extensions, eJFace and eWorkbench. This is the small profile, and is meant to be used on low end devices. More advance devices should use the *Expanded Profile* which features an Expanded eSWT and eUpdate.

8.3.2 Feature coverage of eRCP

This Section compare eRCP with the features list of FoDa as described in Section 2.5 on page 20. Table 8.9 on the next page summarize the features supported by the eRCP in relation to the feature list from Section 2.5 on page 20.

The eRCP is a plugin framework and does not concern it self with with managing connections to networks as well as delivery of content or prefetching of content, this can only be supported if pluings are made that handle this.

The conversion of data is partially covered as it is possible to define content types which enable the engine to convert automatically when detecting the new types but no types are predefined.

Security is partially covered in that it is possible to put restriction os what plugins are able to do, meaning blocking of resources etc. However security in relation to data and its flow is not present aswell as user identification.

A cache is not present in eRCP, but can be made as a plugin. Therefore security and compression of the cache is also not present.

Feature	Supported
Managing connection to a network	no
Delivery of data	no
Automatic conversion of data	yes
Automatic use of best network protocol	no
Handle user identification	no
Handle security	yes
Manage data cache	no
Cache security	no
Cache compression	no
Prefetching of application content	no
Manage system updates	yes
Modular design	yes

Table 8.9: Features supported by the eRCP in relation FoDa

The eRCP does support update features and is modular as everything can be extended or removed in the form of plugins even more than FoDa, because of plugins being handled truly dynamically meaning plugins can be loaded and unloaded at will, which is not possible in the FoDa implementation due to platform restrictions.

The reason the eRCP is such a large framework (byte wise) is that it supports lots of features which we do not require. Table 8.10 list some of the major features.

Feature
Graphics library
Logging
Dynamic events and extensions
Thread scheduling
XML support

Table 8.10: Some of the major features of eRCP

The graphics library enable eRCP to run plugins written for Eclipse directly without modification. This feature is not needed in FoDa as all graphics is handled by the platform (.NET). The logging feature is extensive and enables logging of events and easy debugging. Simple logging is possible in FoDa as used during test to create a trace of execution.

eRCP enable plugins to dynamically subscribe to events and extend extension points. This is not possible in FoDa due to platform restrictions. Dynamic adding plugins are also possible in eRCP, which is not possible in FoDa as all plugins must be registered in the plugin XML file.

eRCP has its own thread scheduler enabling the framework to use the schedul-

ing needed for a certain task, meaning the scheduling can be tailored to specific needs. FoDa is bound to the scheduling of the .NET Compact Framework platform.

Included with eRCP is an XML interpreter which can be used and modified at will. FoDa is merely using .NET Compact Framework XML interpreter as it covers the needs.

Other features are present to make the engine more flexible, such as tracking abilities, possibility to implement ones owns extension registry and the option to undo and redo operations. Theses features are not present in FoDa as they are not needed.

8.3.3 Results

To test the usability of eRCP compared to FoDa, we have produced a simple Hello World application which use the frameworks. The code for the Hello World applications can be found in appendix C and appendix D. To level the playing field we have reduced the FoDa framework to only include the core plugins needed to run a simple Hello World application the same is done for eRCP. Examples of plugins removed cover: DataManager and the Cache from FoDa and from eRCP plugins that contain advanced functionality like the parts of the graphics libraries not needed to display the Hello World has been removed.

The statistics are recorded using the .NET Memory Profiler [39] for FoDa and AppPerfect DevTest4J [40] for eRCP. Both system are profiled on a desktop PC to try and level the base for comparison. The primary reason for the choice of a desktop is due to the fact that the size of the profilers are to large to fit on the mobile devices and made to run on those.

Measurements of memory used do not take into account the memory used by the JVM or CLR to maintain their operation. To get a better comparison of core size, it is measured from the version of eRCP that runs on Windows 2003 Mobile Edition and not the one that run on a desktop PC. Table 8.11 display the gathered statistics of the test.

	eRCP	**FoDa**	Difference
Peak memory used	7.98MB	342.4KB	7.8MB
Memory used	1.5MB	194.8KB	1.3MB
Garbage collection time	60ms	1ms	59ms
Peak objects allocated	49,504	2,072	47,432
Core size	336.7KB∗	16.6KB	320.1KB
Total size	2.74MB	43.6KB	2.6MB

Table 8.11: Statistics of the Hello World applications. *Data acquired from the Windows 2003 Mobile Edition

The statistics show that eRCP use 7.98MB of memory during peak loads and use a memory pool of 1.5MB during normal idle operation, when the application is up and running in comparison FoDa only use 342.4KB during peak loads and 194.8KB during normal idle operation. This indicates that FoDa is less resource intensive than eRCP. This is also clearly reflected in the peak object allocated where eRCP use about 50 times the number of object FoDa use.

The core size of the two systems also differs. FoDa in its reduced state only uses 16.6KB of storage where as eRCP in its reduced state use 336.7KB. This is roughly a factor 20 in difference.

The most notable difference in size is on the total size, which include core, application, xml files etc. Here eRCP requires 2.78MB of storage and FoDa only require 43.6KB. The primary cause of the difference size stems from the graphics library that is required to display the hello world in eRCP.

Conclusion to the test is that FoDa is less resource intensive than eRCP, while still providing the basic features needed to create applications.

8.4 Conclusion

In this Chapter we have performed benchmark of FoDa. We began in Section 8.1 on page 84 by illustrating how much extra work is needed to make use of the functionalities FoDa provide. With the four applications we have used; the "Photoalbum", the "Jump Game", the "Indoor Map" application and the "Pocket 1945" game, we have illustrated that it is possible to make use of the features from FoDa without much extra work. In the application we have tested we had to add roughly 40-50 extra code lines, depending on the simplicity of the application.

After testing how easy FoDa is to use, we have in Section 8.2 on page 101 performed benchmark testing of FoDa to see how much overhead our system will introduce to an application using the system. During the benchmark test we have found that the most significant overhead FoDa introduces is the extra time it takes to start-up an application. Section 8.2.3 on page 103 shows that FoDa will increase the start-up time of an application with roughly 5 seconds. Each extra plugin will increase the overall start-up time with 5.2 millisecond. Section 8.2.3 on page 107 showed that the introduction of new core plugins increase the start-up time even further. If adding less than 130 extra core plugins to FoDa the extra start-up time will roughly be 19 milliseconds extra per plugin added depending on how much code in the plugins that must be executed at start-up. If more than 130 extra core plugins are added to FoDa the formula $g(x) = 0.8896x^2 - 137.95x + 1.5 \cdot 10^4$ should be used instead to calculate the extra start-up time.

The memory tests in section 8.2.1 on page 102 do not explicit show how much memory FoDa introduces to a system. This is because the system uses more memory when it is used, and pictures are shown on the screen. This memory consumption will increase until the Garbage Collector are started, and frees up

memory from pictures which are not shown anymore. This has the result that it will look like FoDa uses more memory when it is used, as seen in section 8.2.1 on page 102 where the difference between Pocket 1945 with and with out FoDa, are greater then the Photoalbum application with and without FoDa. This is because there have been transferred more images in the Pocket 1945 application, then in the Photoalbum application, and the Garbage Collector has not run yet. When we run the Photoalbum application in the profiler .NET Memory Profiler [39], we can see that after each time the Garbage Collector has run, it uses the same amount of memory, as it did when it was started. Since there is never at time in the Pocket 1945 application where it do nothing, and therefore never a time where the used memory amount are stable, do we think the most true answer for the memory overhead of FoDa, is what are introduces on the photoalbum application when using FoDa, therefore is memory overhead of using FoDa 286.4Kb.

Section 8.2.2 on page 103 showed that FoDa consume 112 Kb extra disk space if used in a simple application. However, if used in a larger application the use of FoDa decreases the application size since it is not necessary to bundle graphics and sound with the code, which can be downloaded when needed.

The last benchmark test, described in Section 8.2.4 on page 109, showed that FoDa do not introduce any noticeable overhead in the usage of FoDa features.

Chapter 9

Conclusion

The introduction of this report outlines the focus of this report to develop a system to ease the development of mobile applications for mobile devices. Research showed that this should be attained trough the development of a highly modular and flexible system. To obtain this, selection of technology fell on frameworks and pure plugin architectures technology to support the modularity and flexibility.

Section 8.2.4 on page 109 showed that the developed system is very extendable and modular. Only an overhead of 5.2ms is introduced to the application start-up time with each addition of a new plugin (roughly 19ms if modules are loaded into memory and run upon start-up). Furthermore we tests showed that introduction of more than 130 plugins which are read into memory and run at start-up will cause the garbage collector to start collecting, increasing the start-up time. So number of plugin run at start-up should be limited.

One criterion to consider when developing mobile applications is the limited resources available. The developed framework consider this and Section 8.2.1 on page 102 showed that the use of the FoDa introduces and overhead on memory consumption of 171.9KB when the application is running idle. During usage of the application memory will increase until the garbage collector is awoken at which point memory consumption will return to the level of the idle state.

The primary objective of FoDa is to ease development of mobile applications for the application programmer. Section 2.5 on page 20 presents a list of features which the framework must support as these features will help the development of applications. The features are the result of a number of usecases we find cover a broad spectre of mobile applications. Section 8.1 on page 84 shows the ease with which applications can be written using FoDa. Approximately 50 lines of code are required to use FoDa. For these 50 lines all the features presented in Section 2.5 on page 20 are available to the application. Even porting an already existing application is achieved using no more than 40 lines of modifications most of which are present in two class files. Furthermore the porting of the application resulted in a footprint that was 202.5KB smaller than the original application. This was due to the fact that all content is retrieved from a server and not bundled

with the application.

FoDa even help in the maintenance part of the application an area often neglected by applications as mobile applications are hard to maintain once distributed to the users. FoDa achieves this by supporting patch and content updates delivered from a server, although some limitations to the runtime updating are imposed on the implementation due to restriction of the .NET Compact Framework.

To round of the report we conclude that we have successfully created a framework that makes development of mobile applications easier. We have created a system that fulfils the criteria we set up as well as a system that deliver on-demand content as well as automatic pating of code and content in accordance with the problem defined in the introduction.

Part IV

Appendix

Appendix A

Simulation Thread

```
1  public class DataThread
2  {
3    Vector CurrentLocation = new Vector(0,450);
4    public void run()
5    {
6      WalkTo(20, 453, 2000);
7      while (true)
8      {
9        WalkTo(90, 453, 2000);
10       WalkTo(90, 88, 30000);
11       WalkTo(822, 92, 60000);
12       WalkTo(822, 727, 60000);
13       WalkTo(724, 727, 5000);
14       WalkTo(724, 813, 5000);
15       WalkTo(325, 813, 45000);
16       WalkTo(325, 563, 30000);
17       WalkTo(90, 563, 30000);
18       WalkTo(90, 451, 10000);
19     }
20   }
21
22   private void WalkTo(double ToX, double ToY, double Time)
23   {
24     double StartX = (double)CurrentLocation.X;
25     double StartY = (double)CurrentLocation.Y;
26     for (int i = 0; i < Time; i++)
27     {
28       CurrentLocation.X = Convert.ToInt32(StartX + (i / Time) * (ToX -
             StartX));
29       CurrentLocation.Y = Convert.ToInt32(StartY + (i / Time) * (ToY -
             StartY));
30       if (i \% 100 == 0)
31       {
32         System.Threading.Thread.Sleep(100);
33       }
34     }
35   }
36
37   public Vector getXY()
38   {
39     return CurrentLocation;
40   }
41 }
```

Listing 9.1: “Simulation Thread” for walking around in the computer science department of Aalborg university

```
1  public class Vector
2  {
3    public int X;
4    public int Y;
5
6    public Vector(int x, int y)
7    {
8      X = x;
9      Y = y;
10   }
11
12   public override String ToString()
13   {
14     return (Convert.ToString(X) + "," + Convert.ToString(Y));
15   }
16 }
```

Listing 9.2: Vector class used for the Simulation Thread

Appendix B

Startup log

Time in ms	Log
10	Engine start
141	Check for updates (Starting)
1921	UpdateDLLs (Starting)
1967	UpdateDLLs (Done)
1967	Check for updates (Done)
1969	Initialize ParseXML (Starting)
1987	ParsePluginXML (Starting)
3212	ParsePluginXML (Done)
3212	AssignExtensions (Starting)
3230	AssignExtensions (Done)
3231	Load requires (Starting)
3250	LoadPlugin:RequestManager.dll (Starting)
4046	LoadPlugin:RequestManager.dll (Done)
4046	Load requires (Done)
4047	Load core
4049	LoadPlugin:DataManager.dll (Starting)
4076	LoadPlugin:DataManager.dll (Done)
4077	LoadPlugin:RequestManager.dll (Starting)
4078	LoadPlugin:RequestManager.dll (Done)
4078	LoadPlugin:Photoalbum.exe (Starting)
5272	LoadPlugin:Photoalbum.exe (Done)
5273	Load core (Done)
5273	Initialize ParseXML (Done)
5274	Engine done, starting application
5279	RunCore
5290	LoadPlugin:UpdatePolici.dll (Starting)
5356	LoadPlugin:UpdatePolici.dll (Done)
5361	LoadPlugin:Cache.dll (Starting)
5387	LoadPlugin:Cache.dll (Done)
5387	LoadPlugin:SecurityManager.dll (Starting)
5409	LoadPlugin:SecurityManager.dll (Done)
5550	LoadPlugin:ConnectionManager.dll (Starting)
5580	LoadPlugin:ConnectionManager.dll (Done)

Continued on next page

Table 9.1 – continued from previous page

Time in ms	Log
5585	LoadPlugin:Http.dll (Starting)
5613	LoadPlugin:Http.dll (Done)
5662	LoadPlugin:Prefetch.dll (Starting)
5691	LoadPlugin:Prefetch.dll (Done)
5742	Show GUI
7413	LoadPlugin:Encryption.dll (Starting)
7472	LoadPlugin:Encryption.dll (Done)

Table 9.1: Timetrace log of FoDa Photoalbum startup time

Appendix C

eRCP: Hello World

```
1  package org.eclipse.testercp;
2  import org.eclipse.core.runtime.IPlatformRunnable;
3  import org.eclipse.swt.SWT;
4  import org.eclipse.swt.widgets.Display;
5  import org.eclipse.swt.widgets.Label;
6  import org.eclipse.swt.widgets.Shell;
7  import org.eclipse.ui.PlatformUI;
8
9  public class Hello_world implements IPlatformRunnable {
10
11         public Object run(Object args) throws Exception {
12                 Display display = PlatformUI.createDisplay();
13                 Shell shell = new Shell(display, SWT.CLOSE);
14                 Label label = new Label(shell, SWT.NORMAL);
15                 label.setText("Hello Wordl");
16                 label.setBounds(10, 8, 60, 20);
17                 shell.open();
18             while (!shell.isDisposed()) {
19                 if (!display.readAndDispatch())
20                         display.sleep();
21               }
22               display.dispose();
23               return IPlatformRunnable.EXIT_OK;
24
25         }
26 }
```

Listing 9.3: “Hello World” plugin eRCP, The “Hello world” class

```java
package org.eclipse.testercp;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

	public static final String PLUGIN_ID = "org.eclipse.testercp";

	private static Activator plugin;

	public Activator() {
		plugin = this;
	}

	public void start(BundleContext context) throws Exception {
		super.start(context);
	}

	public void stop(BundleContext context) throws Exception {
		plugin = null;
		super.stop(context);
	}

	public static Activator getDefault() {
		return plugin;
	}

	public static ImageDescriptor getImageDescriptor(String path) {
		return imageDescriptorFromPlugin(PLUGIN_ID, path);
	}
}
```

Listing 9.4: “Hello World” plugin eRCP, The “Activator” class

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id="application"
         point="org.eclipse.core.runtime.applications">
      <application>
         <run
               class="org.eclipse.testercp.Application">
         </run>
      </application>
   </extension>
</plugin>
```

Listing 9.5: “Hello World” plugin eRCP, XML file

Appendix D

FoDa: Hello World

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using IEngine;
using RequestManager;
using System.IO;

public class Photoalbum : System.Windows.Forms.Form, IPlugin
{
    private Label lblName;
    private IEngine.IEngine engine;
    private RequestManager.IRequestManager manager;
    private System.Windows.Forms.MainMenu menuMain;

    public Photoalbum()
    {
        InitializeComponent();
    }

    private void InitializeComponent()
    {
        this.lblName = new System.Windows.Forms.Label();
        this.lblName.Location = new System.Drawing.Point(0, 0);
        this.lblName.Size = new System.Drawing.Size(176, 22);
        this.ClientSize = new System.Drawing.Size(176, 180);
        this.Controls.Add(this.lblName);
    }

    static void Main()
    {

    }

    public void run(IEngine.IEngine engine)
    {
        this.engine = engine;
        this.manager = (RequestManager.IRequestManager)engine.GetPlugin(
            "RequestManager.dll");
        lblName.Text = "Hello World";
        Application.Run(this);
    }
}
```

Listing 9.6: “Hello World” plugin eRCP, XML file

```
<?xml version="1.0"?>
<Plugins>
  <Plugin filename="RequestManager.dll" classname="RequestManager.
      RequestManager" core="yes">
  </Plugin>
  <Plugin filename="Photoalbum.exe" classname="Photoalbum" core="yes">
    <Requires filename="RequestManager.dll" />
  </Plugin>
</Plugins>
```

Listing 9.7: “Hello World” plugin FoDa, XML file

Appendix E

Report Summary

The focus of this report is on the development of a framework for mobile devices which handle on-demand delivery of data as well as automatic patching of code and content. We call the implementing of this framework FoDa.

The report begins with a description of the requirements for the framework derived from a series of use case applications representing the applications most likely to enter the mobile world. The use case applications cover the applications: A Photoalbum, an Indoor Navigation system, a massive multiplayer online game and a karate game with focus on patching the game. The resulting requirements are:

1. Managing connection to a network
2. Delivery of data to the application
3. Handle user identification
4. Handle security
5. Managing data cache
6. Prefetching of application content
7. Managing of system updates
8. Automatic conversion of predefined data
9. Automatic use of the best available network and protocol
10. Cache security
11. Cache compression
12. Modular design
13. Low footprint
14. Platform independent

Using these features we examine six existing system for the ability to support all features. The existing system are:

GameOD
A framework for easing the development of on-demand 3D games.

MOCA
A framework for providing different services.

CAPNET
A context aware middleware system for mobile multimedia applications.

M-commerce
A framework for enabling automatic distributed service discovery for M-commerce applications.

SCaLaDE
A context aware middleware architecture offering informations of the users locations to other applications

PnPAP
A middelware system used to handle all communication with other mobile devices, by use of different P2P protocols.

Most of the requirements are found in atleast one of the system, but not one system covers the entire set of requirements. This indicates that a new system which covers all features are needed.

The design of our system find that the system should be a framework based on a pure plugin architecture as this will allow for high flexibility and modularity of the framework. The design for an engine as well as plugins which cover the features described in the requirements are developed.

An proof of concept implementation of the framework is created and evaluated. The evaluation covers usability as well as performance tests. The usability tests conclude that approximately 50 lines of code is needed for a project to make use of the framework and benefit from the features. One aspect of mobile application development is resource usage, this is tested under the performance tests. Important results of the performance tests are shown below:

Memory use The .NET Compact Framework uses a garbage collector to clean the memory when needed it have not been possible to find FoDa's precise memory footprint due the use of a garbage collector.

Application Size The storage space footprint of FoDa is roughly 112KB

Startup time FoDa adds roughly 5,382ms to the startup of an application and for each plugin an extra 5ms.

Execution Time Result of execution time show that the framework does not slow performance of applications or add to the time it takes to download content.

Bibliography

[1] National IT and Denmark Telecom Agency. Telecom statistics - second half of 2006. 2006.

[2] Sumner Lemon. 500 millioner kinesere har en mobil inden sommer. *Computerworld*, 2007. Last visited: May 2007.

[3] Chetan Sharma. Mobile game conference. 2006.

[4] CipSoft GmbH. Tibia micro edition. http://www.tibiame.com/home/?language=en. Last visited: 06/10/2007.

[5] Microsoft. Microsoft windows update. http://windowsupdate.microsoft.com/. Last visited: 06/10/2007.

[6] Frederick W. B. Li, Rynson W. H. Lau, and Danny Kilis. Gameod: an internet based game-on-demand framework. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 129–136, New York, NY, USA, 2004. ACM Press.

[7] James Beck, Alain Gefflaut, and Nayeem Islam. Moca: a service framework for mobile computing devices. In *MobiDe '99: Proceedings of the 1st ACM international workshop on Data engineering for wireless and mobile access*, pages 62–68, New York, NY, USA, 1999. ACM Press.

[8] Oleg Davidyuk, Jukka Riekki, Ville-Mikko Rautio, and Junzhao Sun. Context-aware middleware for mobile multimedia applications. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 213–220, New York, NY, USA, 2004. ACM Press.

[9] Gary Shih and Simon S. Y. Shim. A service management framework for m-commerce applications. *Mob. Netw. Appl.*, 7(3):199–212, 2002.

[10] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. A mobile computing middleware for location- and context-aware internet data services. *ACM Trans. Inter. Tech.*, 6(4):356–380, 2006.

[11] Erkki Harjula, Mika Ylianttila, Jussi Ala-Kurikka, Jukka Riekki, and Jaakko Sauvola. Plug-and-play application platform: towards mobile peer-to-peer. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 63–69, New York, NY, USA, 2004. ACM Press.

[12] Philip A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39(2):86–98, 1996.

[13] Manuel Roman and Nayeem Islam. Dynamically programmable and reconfigurable middleware services. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 372–396, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[14] Kurt Geihs. Middleware challenges ahead. *Computer*, 34(6):24–31, June 2001.

[15] Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, 1997.

[16] Garry Froehlich, H. James Hoover, and Paul G. Sorenson. Choosing an object-oriented domain framework. *ACM Comput. Surv.*, 32(1es):17, 2000.

[17] Jan Bosch, Peter Molin, Michael Mattsson, and Per Olof Bengtsson. Object-oriented framework-based software development: problems and experiences. *ACM Comput. Surv.*, 32(1es):3, 2000.

[18] Dorian Birsan. On plug-ins and extensible architectures. 2005.

[19] Eclipse. Eclipse. http://www.eclipse.org. Last visited: 06/10/2007.

[20] Mohsen AlSharif, Walter P. Bond, and Turky Al-Otaiby. Assessing the complexity of software architecture. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 98–103, New York, NY, USA, 2004. ACM Press.

[21] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. Proceedings of the Winter 1988 Usenix Conference, 1988.

[22] Azad Bolour. Notes on the eclipse plug-in architecture. *Bolour Computing*, page 27, 2003.

[23] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0 (fourth edition). *W3C*, 2006.

[24] MSDN. Typeforwardedtoattribute members. http://msdn2.microsoft.com/en-us/library/system.runtime.compilerservices.typeforwardedtoattribute_members(VS.80).aspx. Last seen: 06/10/2007.

[25] Microsoft. Designed for windows mobile software application handbook for smartphone. http://download.microsoft.com/download/5/6/8/568a922b-b62c-46d3-a745-0172c2638686/sp_handbook_may2004_final.pdf, 2004. Last visited: April 2007.

[26] Motorola. iden mobile devices. http://www.goteamspeak.com/. Last visited: 06/10/2007.

[27] Symbian. Symbian phones. http://www.symbian.com/phones/index.html. Last visited: 06/10/2007.

[28] Brew. Brew. http://brew.qualcomm.com/brew/en/. Last visited: 06/10/2007.

[29] Nokia. Series 40 platform. http://forum.nokia.com/main/platforms/s40/. Last visited: 06/10/2007.

[30] Plam. Welcome to palm. http://www.palm.com/. Last visited: 06/10/2007.

[31] Wind River. Wind river. http://www.windriver.com/. Last visited: 06/10/2007.

[32] Microsoft. Windows mobile. http://www.microsoft.com/danmark/windowsmobile/default.mspx. Last visited: 06/10/2007.

[33] Jason Zander. Why isn't there an assembly.unload method? http://blogs.msdn.com/jasonz/archive/2004/05/31/145105.aspx. Last seen: 06/10/2007.

[34] Jonas Follesø. Pocket 1945 - a c# .net cf shooter. http://www.codeproject.com/netcf/CfPocket1945.asp. Last seen: 06/10/2007.

[35] Jon Rista. Line counter - writing a visual studio 2005 add-in. http://www.codeproject.com/useritems/LineCounterAddin.asp. Last seen: 06/10/2007.

[36] Dan Fox and Jon Box. Developing well performing .net compact framework applications. http://msdn2.microsoft.com/en-us/library/aa446542.aspx#netcfperf_topic100. Last seen: 06/10/2007.

[37] Jerrold H. Zar. *Biostatistical Analysis*. Prentice Hall International, 1984.

[38] OSGi Alliance. Osgi - the dynamic module system for java. http://www.osgi.org/. Last seen: 06/10/2007.

[39] Scitech Software. .net memory profiler. http://memprofiler.com/. Last visited: 06/10/2007.

[40] AppPerfect. Appperfect devtest4j. http://www.appperfect.com/products/devtest.html. Last visited: 06/10/2007.