Course Report
Fuatara

Abstract
During the autumn -07 a group of 10 students from Uppsala University took a class called project CS (computer systems). This group of students was presented with a system suggestion by Sony Ericsson Research. They called this system CCT or “Converged Community Toolkit” which has later come to be known as Fuatara. Blogging from mobile and receiving notifications on different medium whenever a blog entry is posted are only a few features that Fuatara manages.

Course Review
There were several omissions during the course of the project that caused significant problems later on. In hindsight, these elements are evident; however, they cannot be said to have been easily predictable by an inexperienced crew.

None of the team members were particularly proficient in Web Design (HTML, layouts, et al). Since the project necessitated a web interface, there was significant time spent on hacking HTML to make it work; the end result is a moderately functional interface, but one that is not particularly slick. In addition, none of the team members had any noteworthy experience in graphic design, making it particularly hard to create the all-important graphical elements that make a functional and easy-to-use interface. Instead, we had to rely on an outside source (a friend of a team member with experience in graphic design) for the logos and create our own graphics, taking much time to do something that was not really that outstanding. While it is arguable that the project did not have, as its core focus, a functional and good-looking user interface, that is nonetheless pretty much standard fare in today’s world. We believe that, if we had a person proficient in HTML and graphic design, the project would have progressed faster.

For the majority of the coding, we settled on Java 1.5\textsuperscript{1}. While the language itself was more than sufficient for all our needs during the project, it also comes with a built-in assumption of Object-Oriented coding style. Since everyone indicated having prior experience with the Java programming language, it was generally assumed, by those who conformed to Object-Oriented Programming, that strict object-orientation would be used, as is best practice for Java. This assumption should have been asked (or maybe even tested) early-on in the project so we could have booked a course in Object-Oriented Programming. As it were, the lack of universally agreed-on best-practices for OOP led to some code duplication and re-doing that could have been avoided. This is in addition to the more commonplace refactoring or re-coding done when simply becoming a more proficient coder, or having seen the system’s evolution.

Another major stumbling block was the provided hardware. The low-end computers received were incapable of running Eclipse with Sailfin, or EclipseME, sufficiently fast to not cause frustrations. Small changes in small applications were frequently followed by minutes of compile and deployment

\textsuperscript{1}Java™ 2 Platform Standard Edition version 5.0, if you speak Marketing.
time, before testing the change could commence. Hence, while the computers were sufficient as a test-bed for our packaged applications, they were thoroughly inadequate as a development platform. Since most of the team was unable to provide a computer of their own, this caused chronic delays because of waiting for the computer. This is particularly jarring since the computers provided were relatively new purchases; however, they were Celeron CPUs, as opposed to the slightly more expensive, but much more versatile, Core Duo CPUs.

Project methodology and organization

Scrum Overview

Scrum is an agile development method that focuses on the team’s independence and self support. It is an iterative method which produces a “new” extended product at the end of each iteration. Scrum is a method that allows a lot of freedom in the way of how you integrate it. But there are a few guidelines that we have followed. First there shall be a Product Owner (PO) (which in our case was what traditionally is called the project leader). His job is to be the contact person for the team. The team should be able to work focused and undisturbed during a Sprint (iteration). It is the PO’s job to have a backlog with features in a prioritized order of the system. The team decides how many features they think they can implement from the top of the backlog during one sprint.

Sprint

A Sprint is the time the team decides that they need to be able to implement the next features from the backlog. The PO will also have an estimate of how fast features are implemented (see “Planning meeting” below) and can somewhat make up a time plan of the project when different features will be implemented. This is easy because the scrum masters daily update a “Burndown” chart that shows how much work has been done and how much work that is left during the sprint.

There are not allowed to be any changes of the sprint backlog during a sprint, so if the customer wants to change anything they have to talk to the PO who changes the backlog which will be implemented during the next sprint. This is so that the team will work focused towards a predetermined goal during every sprint. If features are removed or added a new time plan will easily be made by the PO and this way the customer will have an up to date time plan of the project easily at hand.

Planning meeting

At the start of every sprint the team meets up and together decides what goal the next sprint should have. The team also decides what features to implement next and discusses the design of those. All features in the backlog are decided and agreed upon by the PO and customer. The customer may at any time come with suggestions and changes. Any changes that come up during a sprint are not implemented by the team until the next sprint.

Furthermore the team has to decide what approximate points they think every feature will take. This is done by playing cards, e.g. every team member gets a suite of cards with numbers. Then everyone decides a number and puts it on the table in front of him face down. When all cards are on the table
they are turned. Discussions arise if there is a larger discrepancy and the team members argue for their number chosen. This way we see if anybody has misunderstood the feature or even better if someone has a great knowledge or former experience with this kind of feature. The numbers are compared with former similar features and in this way the approximation will be more and more stable during time.

**Daily Standup**

Normally everyone in the team attends a daily standup meeting where everyone in the team should answer the questions

- What did you do yesterday?
- What impediments do/did you have?
- What are you going to do today?

This allows the team to know what everyone else is doing and quickly take action if anything needs to be fixed e.g. design decision or bug fixing. Scrum exists of many meetings but they should be kept small. Normally a daily standup is no more than 15 minutes. It is in the scrum methodology's belief that direct face-to-face communication is the key to fast development and successful software. This is also one of the main tasks for the scrum masters to see to that the team follows.

**End of sprint review**

At the end of every sprint the team meet again and have an end of sprint review where everyone say what they think were good during this sprint and what they want to change. This way the team improves and all members have the possibility to make changes of how they work as a group. Typically all teams meet up with the PO and the customer for a demonstration of the latest product (latest features implemented). This way the costumer will have continuous feedback of how the project evolves and the teams will easily receive feedback from the costumer.

**Scrum – how we used it**

**Why scrum**

Scrum was introduced to some of the students during a class in software engineering at Uppsala University. Since none in the project had former experience with working with a project methodology we chose scrum because it’s an iterative developing method which seemed to suite us well because we frequently had an extended working version of the system which we could demonstrate and get feedback on from our customer (Ericsson Research).

**Implementation**

We started out as one large scrum team with 10 students and one scrum master. Normal a scrum team exists of 3 to 8 persons. We read through the agile manifesto and “Scrum and Xp from the Trenches” by Henrik Kniberg and started to implement our version. Daily meetings, sprints, backlog with features to implement and a scrum board.
Organizational changes

After having the possibility to attend a seminar about scrum with Henrik Kniberg and Jeff Sutherland, one of the “co authors” of scrum, we decided to restructure a bit. Our scrum master at that time discussed with Jeff Sutherland and he suggested that we should split up our team into smaller groups to improve productivity. It should be possible to increase your productivity about 2 times according to Jeff Sutherland who had some figures from a company that had done some studies and tried this. He said that it later has been more common with smaller scrum teams with 3-4 persons rather than 7-8. With this in mind we decided to split our team into two groups of 5 students each.

Two new scrum masters were assigned from each team and the former scrum master became our PO (process owner). Since the workload on this position wasn’t comparable to the other development positions the PO also became one of the team members in one of the teams, resulting in that one of the team existed of 1 PO, 1 Scrum master and 3 other students. Not the optimal structure with a PO within the scrum team but it has worked well for us and the PO could attend all meetings since we chose to have the two team sprints interleaved.

The main purpose for the scrum masters was to glue the group together and make the team work in the same direction. One team focused on SIP/XMPP IM communications as well as on the mobile client. The other team developed the server structure for notifications as well as the web managing system. The two scrum masters together with the PO answered for the integration between the two teams so that there were no design issues or other team collisions.
Retrospective

The common feeling is that scrum has really been a help within this project. When we look back at other project reports from earlier years we can see that they still hadn’t start coding in early November where we started of only a few weeks after the project launch. We had a basic version up very early and we think that it has been a great value to us since the project existed of a lot of different parts that were to be assembled into one system. It has been much easier to develop the system correct from the start since we have been able to give and receive feedback both from and to our customer throughout the entire project.

The daily standup meeting has also been a grand help in the sense that impediments and problems have easily been detected and a quick solution has been decided upon by the team. One example to this our standup meetings started with one of the team members picking up a token (a blue whiteboard pen in our case) and presented his part, followed by handing over the token to any person of his choice. The problem was that no one wanted to start talking. The suggestion from one team member were that anyone could pick up the token at the start of the meeting and directly hand it over to another member who had to start talking. Now it suddenly became very interesting to get the token first and give it away. No more silence in the beginning of the meetings.

The reason for not having the scrum master staring to talk every time is because the team should work as a team and the scrum master is not the boss over the other in that sense. In addition the team members should not talk and explain to the scrum master; rather, they should be talking to the whole team.
Product Report

Fuatara

Abstract

During the autumn -07 a group of 10 students from Uppsala University took a class called Project CS (computer systems). This group of students was presented with a system suggestion by Ericsson Research. They called this system CCT or “Converged Community Toolkit” which has later come to be known as Fuatara. Blogging from mobile and receiving notifications on different medium whenever a blog entry is posted are only a few features that Fuatara manages. An extensive description of the system can be found under the Fuatara heading. This report will describe the system in depth and take up issues and solutions during the development of the product.

Introduction

Original Product Proposal

- Read a blog entry from mobile
- Specially format blog for different screen properties
- Post a blog entry from mobile
- User registration
- Add people to buddy list
- Edit buddy list
- Send blog update notification to mobile
- Desktop Notification Widget
- Notification on RSS Feed
- "Presence Service" (See people online)
- Send message mobile -> desktop
- Send message desktop -> mobile
- Send a photo to blog from mobile
- Online presence (Away, Available, ...)
- Send/Receive IM to & from ICQ/MSN/AIM
- Chat with several people ("Chat Room")
- Updates via email
- Updates via SMS
- Updates via MMS
- Visitor tracking of blog
- Rich Text Notifications
- Voice blogging
- Video blogging
- Location-based "Friend Finder"
- Buddy list categories
- Preliminary SIP Research
- Site Admin UI
Final Product – key features

Blog:

- Add a blog
- Edit/remove blog
- Publish a post on the blog
- Send blog update information to servers
- Get mobile screen size when mobile access blog

CCT Webpage:

- Register/unregister a new user on webpage
- Login webpage
- Edit user’s profile
- Add blogs/rss feeds on webpage
- Subscribe to blogs and rss feeds
- Search blogs and rss feeds
- List popular blogs and rss feeds.(“popular” means many users subscribe to the blog or rss feed)
- Admin manages users’ blogs and rss feeds
- View buddy list of jabber client on webpage
- Add buddy to buddy list of jabber client on webpage
- Fetch buddy list from jabber server
- Webpage Tutorial and contact admin

Fuatara client:

- Login Fuatara client
- Capture images, audio and video from the mobile
- Post a blog post with text, image, audio and video
- Add a user to a buddy list
- Manage buddy list
- Chat with other buddies who may use another IM with different protocols
- Send offline message to buddy/buddies
- Show/change presence

Servers:

- Send Email, IM ,Widget notifications to users whose subscribed blogs/rss feeds get updated
- Send IM notifications to mobile user according to the mobile’s screen size
- Jabber server
- SIP server
- Gateway for XMPP/SIP

Target phones/platforms

- Sony Ericsson K530i JP7 176x220
- Sony Ericsson W880i JP7 240x320
Preliminaries

Important concepts

- **Widget** - Widget here in our project is a tool to receive notifications update information and use Snarl to show the notification. It runs in background and receives notifications from server.

- **Snarl** - The notifications received by the widget are displayed by the 3rd party application Snarl. A brief notification and a link to the update are displayed in a user configurable pop up on screen. Clicking the pop up opens the link in the browser.

- **Filter** - Add a filter string for notifications. There are exclusive and non-exclusive filters. When using an exclusive filter, program disregards those notifications that include the filter string i.e. only when the update does not include the filter string, program will send notification of the update to users. For non-exclusive filter, program disregards those exclude filter string.

Glossary

- **Blog** - A web log displaying entries in reverse chronological order.
- **Buddies** - IM friends.
- **Buddy List** - The IM contact list.
- **CCT** - Converged Community Toolkit.
- **CPIM** - Common Presence and Instant Messaging format
- **Friend** - A user that another user has given permission to view their (the second user's) private blogs.
- **IM** - Instant Message.
- **"Image Notification"** - A notification with an attached image.
- **JabberWookie** - A Java Jabber communication library
- **MMS** - Multimedia Messaging Service. A standard for sending messages that can contain more than just text.
- **PO** - Process Owner (in Scrum), normally the project leader.
- **Private Blog** - A blog which can only be viewed by users who are currently logged in and who the blog's owner has listed as friends.
- **Protected blog** - A blog which can only be viewed by users who are currently logged in.
- **Scrum** - An Agile development methodology.
- **SIP** - Session Initiation Protocol. Used for negotiating peer-to-peer media sessions between two or more users.
- **SMACK** - A Java Jabber communication library
- **SMS** - Short Messaging Service. A standard for sending text messages to phones.
- **Spark** - An open source instant messaging program for desktop computers
- **Sprint** - Time period for one developing iteration (two weeks normal during our project).
- **Widget** - A component of a graphical user interface.
- **XML-RPC** - A spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It's remote procedure calling using HTTP as the transport and XML as the encoding.
- **Atom** - Atom is an up-and-coming format for editing, syndicating, and archiving weblogs and other episodic web sites.
Tools used

- Eclipse/Eclipse ME. Eclipse is an open source software framework.
- Windows (XP, Vista)/Linux Ubuntu/Mac OS 10
- Sony Ericsson J2ME developer tools
- Apache 2/MySQL/PHP
- XAMPP. One click install developer package consisting of Apache, MySQL, PHP and Perl.
- Wordpress. Wordpress is an open source blog publishing system written in PHP and use MySQL for database. It supports extensible plug-ins.
- OpenFire. OpenFire has the most features, GPL license and has easy administration from a web GUI.
- CodeIgniter. CodeIgniter is an open source PHP web application framework with small footprint and rich set of libraries.
- JDK 6
- Wiki. Wiki is software that allows users to create, edit, and link web pages easily.
- Tortoise SVN. A Subversion client, implemented as a windows shell extension for revision control / version control / source control.
- CruiseControl. A Java-based framework for a continuous build process.
- Bugzilla > Trac. Bugzilla is a Web-based general-purpose bug tracker tool; Trac is an open source, web-based project management and bug-tracking tool.
- Snarl. Snarl is an application that lets our widget display popup window notifications.
- Wireshark. Network protocol analyzer that runs on Windows, Linux and OSX.
Fuatara

Overview

Fuatara is a system, including a Fuatara webpage and Fuatara client software with services like IM on mobile phone, and a Wordpress weblog website. On the Fuatara webpage, people can register as a new user and subscribe to blogs and rss feeds. Users may also use the same login details for the Fuatara mobile client and get notifications when subscribed blogs and rss feeds are updated. Notifications are also available as email and on our desktop widget. Mobile Fuatara client is not only used for receiving notifications, it also can be used for chatting with other IM users who perhaps are using different IM clients with different protocols like MSN, ICQ, SIP and AIM. Furthermore the Fuatara client can be used for blog posting with pictures, audio and video captured on your mobile.

Blog

The Blog plug-in sends notifications to the Fuatara server when a new entry is posted on the blog. We use Wordpress as our weblog website for users to publish posts which may include text, pictures, audio and video. It also has the ability for readers to leave comments.

![Fuatara Dev Blog](image)

Figure 1: Fuatara dev blog

Users also can manage their own blog after signing in as admin. It is possible for blog owner to

- add, edit and delete their posts,
Figure 2: Edit of entries

- manage blog the plug-in, which include inactive and deactivated plug-ins,

Figure 3: Manage Blog plugin

- Manage more settings; include the blog theme, general options etc.
RSS

On our Fuatara web page, Users can subscribe to some rss feeds which they are interested in. Once a feed was subscribed to, users will get Email, IM, and Widget notifications (it’s up to the user to decide what kind of notification she/he want) when the feed gets updated.

Notifications

When users subscribes to blogs or add rss feeds on our Fuatara web page, once these blogs or feeds get updated users will receive notifications. These notifications can be sent to user by Email, IM and Widget. However these notifications are just brief descriptions of update and a link to the webpage updated.
**Mobile client**

Fuatara also has its own downloadable mobile application. Using Fuatara on the mobile it is among other things possible to do blog posting with video and audio. Fuatara on the mobile also lets you send messages to your friends, not only at Fuatara, but also on MSN, ICQ, SIP and AIM.

<table>
<thead>
<tr>
<th>Notification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
</tr>
<tr>
<td>[Vlad's blog]: Mars</td>
</tr>
<tr>
<td><strong>Content</strong></td>
</tr>
<tr>
<td>4th planet: &lt;image&gt;</td>
</tr>
<tr>
<td><strong>Url</strong></td>
</tr>
<tr>
<td>Read more: <a href="http://acrose/wordpress">http://acrose/wordpress</a></td>
</tr>
<tr>
<td><strong>Image:</strong></td>
</tr>
<tr>
<td>[Image]</td>
</tr>
</tbody>
</table>

**Figure 6: Mobile client**

Notifications are sent directly to your mobile and include a link so you can quickly access the blog page.
Design

Overview (notification flowchart)

![Notification Flowchart]

Figure 7: Overview (notification flowchart)

The Input server is the heart of the notification system. It receives blog entries from the blog notification plugin (installed on Wordpress) and updated rss feeds. The Input server than connects to our database and checks for authentication information like username and password. It also fetches all users who has subscribed to the specific blog/rss feed. The list together with the post entry will be sent along to the Dispatcher which has the single task of distribution of post entries together with the user who has subscribed for e.g. Email notifications. The task for the format modules are as the name suggests reformating the post entry to suit the targeted receiver.

Design decisions

Blog platform

Wordpress was chosen because of extensibility, community support, popularity and compatibility with the rest of the Fuatara system (Apache, MySQL and PHP).

RSS Feed Aggregation

The decision to implement RSS feed aggregation was based on the need for Fuatara to fulfill some purpose even under the assumption that none of our custom, and hence proprietary, solutions are in use. For example, for a standard Fuatara notification, you either have to use WordPress and our plugin, or support the Fuatara protocol. This is a barrier for adoption. On the other hand, most blogs (and many non-blog sites) supply an RSS feed, which allows us to use a parser to check for new updates.
A separate server handles RSS Aggregation in Fuatara: the RSS Reader. Its job is to read the feeds subscribed by users, and generate notifications when an update has occurred. Once a notification is generated, it can propagate through Fuatara using much the same mechanisms as a regular notification. In fact, past the Input Server, there is absolutely no distinction between a notification from a feed and that from a blog.

There are two main draws of Fuatara’s RSS notifications. One of the advantages is mirrored from services such as FeedBurner and Google homepage, in that multiple people subscribing to the same feed only poll the source once. This is a draw for server owners/publishers, because they do not need to pay for the additional bandwidth, their page loads faster, and their host is happier. The other is that the Fuatara delivery method, which reaches across the desktop and mobile space, is primarily a push delivery method, meaning that it costs a Fuatara service less bandwidth to deliver the information to end-users than it does FeedBurner or Google. In addition, users get messages from Fuatara faster than any other service.

**XMPP/SIP**

The protocols used for IM are XMPP and SIP. Those protocols were part of the original project proposal which called for XMPP on the mobile and a SIP enabled client for the desktop.

**Java 1.5**

The majority of the project’s coding was done in Java 1.5. The main rationale behind this was that Java is, despite all drawbacks, a more-or-less write-once, run-everywhere language. Since we had no requirements or goals to make a high-performance system, the language’s main drawback – that of running everything in a Virtual Machine, causing a certain amount of slowdown, and the necessity of running a garbage collector – could be accepted.

Another main reason for the choice of Java as the base programming language is its ubiquity. Java is a widely used language at the moment, and is only gaining traction. By using Java as a base for our servers, we were able to make use of publicly available, Open Source, libraries such as ROME, Propono, Jain and Smack to quickly develop the interfaces to the different protocols, leaving us with more time to focus on the mainstay of our task of converting the data from one to the other.

The usage of Java 1.5 has allowed us to develop and deploy the Java portions of Fuatara on Windows, Linux and Mac OS X. We selected the older 1.5 version of Java in order to have a broader compatibility than provided by Java 1.6, which is still in the process of uptake, and had only been available for seven months at the beginning of the project. There are, however, no indications that Fuatara is code incompatible with Java 1.6, so when Java 1.6 achieves higher adoption it is possible to migrate to 1.6 and take part of the improved JVM.

In retrospect, the use of Java served the project very well. Thanks to excellent tools, such as Eclipse, it has been miraculously easy to create at least passable Java code. With Java we have also been able to avoid errors and pointless crashes on the basis of memory management and incorrect pointers; items that are a major problem in C++ et al, and would have greatly impacted us if we had chosen such a language.
**CodeIgniter**

CodeIgniter is a PHP framework that provides a certain set of functionality for PHP scripts. It is based on an object-oriented approach to coding, forcing the separation of database access (“Model”), computational part (“controller”) and user interface (“view”). CodeIgniter is free and open source software, allowing us to make use of it immediately and without any issues.

CodeIgniter was chosen because it is a relatively small framework, compared to the others available, and simple to use and modify. The object-oriented approach also allowed us to re-use ideas and methodologies used in our Java applications, making us liable to commit fewer mistakes due to mixing ideologies. The alternative (not using any framework at all) would have required significant time invested for minimal gains on the website front. CodeIgniter also has the base assumptions of default-deny, and we have tried to implement that wherever feasible in our code. While not explicitly tested, as it was neither requested nor required, that should cover at least the less esoteric security breaches.

Just like using Java, the Object-Oriented methodology of CodeIgniter did not sit well with people who were not used to writing object-oriented code. This posed a barrier in the first few weeks of working on the project, but one that gradually began to lessen as the project continued. With everyone up to speed on how to use CodeIgniter, we were able to produce good code with a very good speed.

**Componentization**

The back-end servers of Fuatara are divided into 9 logical components. The design is based on the modular paradigm, where a system is based upon a collection of loosely-based components that communicate through a specified protocol. This allows each component to be replaced with one that does an equivalent task, but in a different manner. For example, any given format server could be replaced with a load balancer that pretends to be that format server. That load balancer can then split up the workload to several different physical servers. This is possible without modifying the dispatcher or the format servers themselves.

This architecture allows the system to attain scalability via parallelism: if more throughput is required, adding additional servers is easily possible. However, for a single-server setup, the loose communication (TCP sockets and serialised protocol) requires more computational power on each end to encode and decode the data. A tight approach would yield more performance for an individual box. No matter how much performance one can extract from a single box, however, it is still highly inferior to parallelism, as has been proven by supercomputers being clusters as opposed to mainframes. Therefore, we allow improved scalability at the cost of individual efficiency.

The division into components is based on the Object-Oriented methodology. Each component has a particular task that it does well. Any data processing that is tight, i.e. requires comparison of data only needed for the processing itself, is a part of that component. Subsequent data processing that acts only (or to a major part) on the resulting data can be extracted into a separate component, and this is done wherever we have had the time.
**cctlib**

The cctlib archive is a collection of utilities to make it easier to create the individual servers of Fuatara. It contains almost the entire networking stack (including data models), creating an almost automated way to send data from one process to another via the network. In addition, it includes a host of utilities used, such as a logging utility, automated configuration access, Base64 encoding, database access and other utilities.

Data structures are passed between servers as basic data type to preserve OO. cctlib allows a server or client to be created quickly by supplying:

- Network stack implementations (XML Stream listeners and senders are available that can immediately read any and all XML objects used by CCT Notifications)
- Logging & Configuration support, using standard config & log files for CCT
- Database access, where needed

We were essentially forced to create cctlib once we had hit 4 individual components. At that point, each server had individual implementations of everything it needed to do, be it configuration or networking. Since we relied heavily on sending data from one server to another, this started to create a divergence in the protocol between the newer servers and the older ones, which weren’t being re-coded. At that point, we had the choice between always re-coding the servers, and unifying them. The logical choice was to unify all the servers’ networking code into a central library, and allow the individual servers to make extensions where necessary.

Since it was created, cctlib allowed us to continue expanding Fuatara at a much more rapid pace. Not only does it reduce the amount of duplicated code in existing servers, at its current stage of development it severely lowers the barrier for creating new servers and extending the protocol (no longer requiring code modifications to add new elements, for example; subclasses are now able to do this).

**InputServer**

InputServer is the main server run loop for the server application. It's responsible for receiving connections from input sources, and launching request handler for each incoming connection. It handles connections' authentication and fetching subscriber data from database, and dispatches the different types of notifications to different format servers which include EmailFormatServer, IMFormatServer and ClientFormatServer. When an input module connects to the input server, it must authenticate itself via a timed authentication. The input server then checks database for all subscribers to that notification.

InputServer is the first step in sending notification. The input part of InputServer is, in effect, a notification of itself. It notifies the server of an update, and it is the job of InputServer to filter out inappropriate notifications.

Of all the servers, InputServer has by far the most interactions with the database. It must check if the notification it received is from an authorized source (a registered blog or another server). Once the notification clears authentication, it has to fetch all appropriate information about subscriptions from the database. However, once it has fetched all the information, all it needs to do is forward the data to the Dispatcher for processing.
We initially intended InputServer to be just the authentication portal for Fuatara, handling only authentication before passing on the request. However, we later decided to add the data fetching part to InputServer as well, even though this might be better suited for Dispatcher, since InputServer already maintained a database connection.

**Dispatcher**

Initially, Dispatcher was envisioned to receive prepared notifications complete with pre-processed recipients (i.e. filters applied) then forward these to an array of different format servers. In effect, it was supposed to be a load balancer and a separator. However, once it became clear that most of our servers are stateless, and thus wholly replicable, hardware-based load-balancers would suffice.

Dispatcher was instead morphed into becoming wholly a data separator. It has acquired the ability to apply filters on data, in addition to determining which format server a notification should go to. This version of Dispatcher evolved from InputServer, so it is not fully separated from its parent. While we envision Dispatcher as a logically separate entity from InputServer, we have not completed the separation yet, though not much remains. The interaction between the two has been narrowed down to one function call.

**Format Servers**

From the beginning, we assumed that the major workload on the system would fall on the format servers, as these would have to be responsible to sending the data to the clients. A single notification going into a format server would potentially have to be replicated thousands of times by a single server. To more easily accommodate expansion, we decided that each different data output module should be logically separated from the others. Each one of those is a Format Server, taking as its input a notification and being able to do two things.

First, each format server would receive the full notification, which usually includes more text than what is shown in the final version. It is then the format server’s job to cut down on the text, and strip content, such as to make it appropriate for viewing on the recipient’s medium. Each format server has generally different format needs, making them overlap very little in the formatting functionality.

Once the message has been properly formatted, it is each server’s job to send it to all of the recipients it is supposed to reach. The format servers are not obliged to save messages; only to send them. For email and IM this is generally okay because the formats are not lossy. For Fuatara clients, it is defined as okay, seeing as the client keeps no history, it is regarded as a “currently, at this moment”-scenario.

**EmailFormatServer**

EmailFormatServer start loop for connections with multi-thread. Once it accepts a new connection, it requests if the connection is received correctly (to verify this EmailFormatServer sends a reply to the client with a current timestamp, then proceeds to compute the MD5 checksum we expect to find in the reply from the client). After the authentication it constructs an email according to the data that was fetched from the InputServer, and sends this email to all the users who subscribe to the updated blog.
The email format server is the most straightforward of all the format servers. For security issues, it strips messages of all their HTML code before they are sent to the reader. That is everything email format server needs to do.

We considered creating a more complicated email format server, capable of processing HTML emails and sending them, with the rationale that people were using clients that support the display of rich text messages; however, this was discarded as a lot of work. It is still a valid future development. The Email Format Server was developed as a replacement of sorts to the SMS gateway we discarded. By using an operator-provided email address, the user is nevertheless able to receive an SMS upon an email notification, which we considered an adequate substitute.

**IMFormatServer**

The IMFS is our gateway to Jabber, sending IMs to clients. It is also the primary gateway to mobile phones. The IMFS has more tasks than any other format server when it comes to formatting. Just like CFS, IMFS has to strip HTML and other content from the messages it sends out. In addition, however, the IMFS has to (in the case of mobile clients) make it possible for clients to receive image notifications, by resizing image notifications to fit the mobile screen.

IMFS sends image notifications by first detecting if there is a mobile recipient. If there is a mobile recipient, it downloads the attached image to a Fuatara-hosted image storage. Then it resizes the image to the screen size of each client (each Fuatara mobile client, upon login, reports its screen size to the IMFS), and that image, as well, is stored on the Fuatara-hosted image storage. It then sends the notification, coupled with image, to the mobile client.

While this may make it sound like the IMFS is not replicable, that is not technically true. With only a minor modification to the IMFS base logic – to make each server connect to the Jabber server with a unique resource handle – it is possible for the IMFS to replicate just like the Email FS. This is because Jabber will, in that case, send the mobile client’s message data to each Email FS, thus replicating the data across all servers. While this solution is not optimal, it is trivial; a better one is to create a database that matches users’ last reported screen sizes to usernames. This would allow formatting servers that do not know a user’s screen size to check it from the database.

**ClientFormatServer**

The CFS is responsible for accepting connections from Fuatara clients, then sending notifications to the clients once they are connected. It must also strip any HTML from the notifications, and cut them down to a reasonable size (the clients will not display more than 3-4 lines of text).

The CFS is unique in that it accepts connections from two sources. It accepts notifications from internal Fuatara servers, and connections from Fuatara clients. For the latter, it needs to authenticate the users against the login database, making it the only format server to require database access. While the CFS handles a list of clients, giving it a “state” of sorts, each of those states have no bearing on the global action of the server. Thus the CFS can be replicated by using two load balancers. On the client side, each client would, through the load balancer, connect to only one CFS. On the server side, a custom, Fuatara-capable, load balancer would connect to every CFS and send the notification to all CFSs.
The inclusion of a CFS was necessary to facilitate the Fuatara Client; however, that is by no means the only purpose of the CFS. What it does is, essentially, pipe notifications from one Fuatara instance to another. Using this method, it is possible to, with appropriate receiving end, connect two instances of Fuatara together so one receives notifications from the other. It is also possible for third parties to provide services based upon Fuatara notifications, by connecting to the CFS, essentially making it a loose API.

**Fuatara Client**

The desktop client (formerly “widget”) is the product of needing a more desirable interface on the computer. Receiving IM notifications on the computer was ugly and wasted much space, and readability. Email notifications, while functional, are not nearly as attention grabbing as would be required of the type of notifications one would want. Thus we settled on making a client for the desktop with a specific interface tailored to receiving messages.

We have implemented the interface using two OS-native libraries: Snarl on Windows and Growl on Mac OS X. These allow us to display our notifications in tandem with other applications, with the user having full abilities to modify the appearance of said notifications. The support for the native libraries is achieved using JNI, making the source for Mac OS X and Windows essentially the same, with only the JNI library being affected by the changed operating system.

The use of Java for the Fuatara client is justified by the investment needed in creating it. The Fuatara Client is an example of using cctlib to extend Fuatara from an off-site location. The Client uses less than 100 lines of code to interface with Fuatara, with the rest being able to focus on its task at hand of providing notifications. It was thus easily justifiable for us to create the client (as it was not much work), but it also stands as a showcase of how to expand upon Fuatara.

**PostServer**

PostServer allows users to, from their mobile phones, submit messages for posting – through Fuatara – to any Atom 1.0 compliant weblog. This gives it two obvious functions.

One is to receive data from the client. PostServer is able to receive text data for the publishing via Jabber. An XML message, sent from the mobile client, specifies the text data and properties of the post. That message may, optionally, also specify a request to attach a file to the blog post. PostServer will then provide, again via Jabber, an XML reply to the client specifying how to upload the data, which may be any supported file format.

PostServer opens a port and listens to the connections that come from mobile clients. When receiving a connection from a client, the first step taken by the PostServer is to identify the connection. Next, it needs to authenticate the connection by checking the password. The first packet sent from mobile just contains some basic information of the post. After identification and authentication, PostServer receives file from the client and then check the file type, uses Atom to upload the media file to the user's blog according to its file type. If the post does not contain any file, i.e. the post just contains text, post it along without receiving file or get related data from database.
When PostServer has received all post data, including attachment if any, it will attempt to connect to a previously specified (via the Fuatara website) weblog for publishing. If the weblog supports Atom 1.0, it will be able to create a new post with the appropriate attachment.

PostServer differs from most other servers as it is a highly separate part, and it doesn’t even deal with notifications to begin with. However, it was necessary to implement this functionality, as no other server could possibly be adapted to support upstream, as opposed to downstream, data. PostServer is the third iteration, being the sole cause for our switching around XML RPC and Atom. The modular approach, however, meant we lost only as much time as it took to re-code the actual blog publishing module, which we consider a success of the Object-Oriented methodology used.

**RSSReader**

The RSS Reader does exactly what its name implies, it reads RSS feeds. When the RSS Reader detects a change in the feed, its job is to create a Fuatara notification, which it sends to InputServer, for further processing.

The server’s main thread wakes up every 15 minutes. Once it wakes up, it dispatches a new thread to handle RSS reading. This new thread handles all subsequent processing independent of the main thread. Several processing threads can co-exist. Since the weblogs are always the same list for each thread, there is effectively 15 minutes time lapse between each thread, which means that, assuming statistically equal access times, there’s approximately 15 minutes between each weblog check.

Once it wakes, the first task of the RSS reader is to check all RSS feeds in the database that have at least one subscriber. This prevents us from crawling unpopular feeds. The feed data is then passed along to a crawler.

When the feed data is received, the main loop starts. Using ROME, each feed is downloaded and parsed. Once downloaded, the feed is checked for updates by first checking its timestamp and, if timestamps don’t exist, checking the text contents of the first post.

When a feed has updated, its latest entry is sent along to the Input Server for processing.

Initially, we wanted to include an RSS reader because we thought it should have been a relatively simple modification: just send a notification through the system and it should leverage the existing infrastructure. The RSS support we ended up having was necessitated by our inability to implement everything in a single database structure. This has highlighted an inflexible initial database system, one that we have tried to alleviate but is still an avenue for improvement of Fuatara.

Our first expectations were that, to implement RSS support, we needed to have routines for it in the website, as part of the interface, and the RSS Reader needed to be added. The final product is not too far from those expectations: the additional modification needed was to give InputServer knowledge of the RSS database to fetch data from there. This data can subsequently be made to fit the protocol, so components past InputServer – including Dispatcher – are entirely unaware that the data source is a feed and not a blog.
**Desktop SIP Client**

We have not developed our own client for the desktop side. Instead we have used SipCommunicator while developing and testing. This client was chosen because we could quite early in the project communicate with it with the experimental SIP code we had setup. In addition we could get a consistent behavior from Wireshark printouts that seemed to comply with the SIP RFC documentation we had access to.

Our goal was to provide support for a SIP client in general. However, as the system in practice only has been tested with SipCommunicator we have very little knowledge about how it cooperates with other SIP clients.

The use of SipCommunicator has not been without problems. At least one time the discussion was raised to replace it with a client of our own making. This would have increased the chances of perfect interworking with our system. The client chosen has its faults, and the authors described it as experimental and that it may not always work. However, if we had made our own client we faced the risk of producing another unstable SIP client, when working with other systems. As problems resolved we decided to keep SipCommunicator. The team had invested a lot of time to understand its behavior through testing and Wireshark testing so it seemed worthwhile to keep that investment.

**Gateway**

**Overview**

The gateway is responsible for communication between SIP/SIMPLE and XMPP. We have been able to achieve a gateway that can handle both messages and presence between the mobile Jabber client and the SIP part on the desktop side. The design of the gateway has been towards extensibility for other protocol conversions rather than for performance. Our gateway operates as a component to a Jabber server. This means it does not use a true server to server connection but instead relies on the Jabber server being able to handle external components. We originally wanted a server to server connection but were unable to achieve that in reasonable time. This external component mode is described by the Jabber standard so, in theory, every Jabber compliant server should be able to work with our gateway. But, the main drawback with this mode of operation is that we got a less secure connection.

It consists of four main parts:

1. SIP/SIMPLE Module
2. XMPP Module
3. Transformer Module
4. Deferred Messages Module

The gateway has XML configuration files for such settings as server address, ports, and protocols used and the transformer classes that should be used.

**SIP/SIMPLE**

The SIP/SIMPLE Module is responsible for the SIP Layer's routines, which in charge of instant messaging and presence transmitting with the SIP Server. The XMPP messages and presence information are passed on to it through the CPIMtoSIP transformer class. We use the open source
library JainSIP to establish the connection between SIP Server and the gateway as well as maintain dialogs between clients. SIP deferred message is implemented in SIP Layer.

The major problem we encountered during development was to track the SIP package flow. The sip protocol specification RFC3261 is not a complete one and the actual implementation may vary in different SIP clients. When we realized clients and servers did not follow the same package flow standards we were in a position where we always had something to do. These, non standard behaviors were a big problem for our development. In order to make a clear picture of the package flow, we used Wireshark to catch packages and do analysis. This includes a lot of tough work. Here the challenge was to make sure it worked with our selected desktop client and, more importantly, at the same time not prevent it from work with other SIP clients. We were trying to find a match between SipCommunicator and our gateway without becoming too dependent on only SipCommunicator.

The main tasks of SIP/SIMPLE Module are:

1. Receive packets from SIP Server
2. Response (Send response packets) to SIP Server
3. Pass packets to Transformer Module
4. Create SIP/SIMPLE packets from the output of Transformer Module
5. Send packets to SIP Server

The present SIP/SIMPLE module in the gateway has gone through some iterations, the most important one being the switch to Jain SIP. This as a part of our replacement of the first gateway made for Sailfin. With this switch we were able, within a few days, to get a first primitive gateway up and running. The switch from Sailfin also had system design implications, which in hindsight, come to direct some of the future work. Previously we had servlet code running on a server, and now we had an executable jar file with a client to server connection. This was deemed unsatisfying and work started getting the gateway to behave as a server. Which in turn created more problems and in the end the solution was to let the gateway be a component to Open Fire.

**XMPP**

The XMPP Module takes control of the XMPP Layer’s routines, including instant messaging and presence communication with the Jabber Server. The Module receive, send and forward packages and store sessions between users. The XMPP Layer of our gateway is using component to server connection to a Jabber Server. This means that it is declares as an external component in OpenFire, our Jabber server. We originally wanted a true server to server connection between our gateway and the Jabber Server, but were unable to achieve that in reasonable time. Here we tried to use the connection parts from the Spark clients’ code and find a library that would allow us to get this working.

This module performs seven tasks:

1. Startup connections to the Jabber Server specified in the configuration file
2. Periodically send request to all the Jabber Server to see if they are still there
3. Listening for packages from Jabber Server
4. Forward packages to Transformer Module
5. Receive packages from Transformer Module
6. Send packages to Jabber Server
7. Maintain all sessions which are still used for interaction between two users or servers

This module uses the JabberWookie library for communication with the Jabber server. This library replaced the SMACK library, which we previously had been using, since we could not get a component to server connection with SMACK. This works fine except you must add the component after the @ token when specifying a SIP contact from the Jabber client, e.g.

Joe%130.238.15.169@sip.Fuatara.com.

Packages are classified as Presence or Message, the IQ and Chunk (parent class of all other) types are currently not handled. The Presence package includes the following types: subscribe, subscribed, available, unavailable, and status. The sending and receiving of Jabber packages are done by the class XmppComponentLayer.

**Transformer**

The Transformer Module takes charge of message and presence transformation between SIP/SIMPLE and XMPP. It is a set of classes that handles the transformation from one protocol to one another. Currently we are able to handle the conversion between SIP and XMPP.

The central piece in the design is the use of an intermediate format. All messages and presence information are first converted into the CPIM format.

For the transformation we have made four classes:

1. SIPtoCPIMtransformer
2. CPIMtoSIPtransformer
3. XMPPtoCPIMtransformer
4. CPIMtoXMPPtransformer

These are all registered in the file transformers.xml and all future transformers will also need to be named in this file. The gateway uses this xml configuration file to get the class names of the actual transformer to use. The reading of and construction of XMPP messages are done with the JabberWookie library.

The decision to go for an intermediate format was based the prospect multiple protocol support would be easier to implement in the future. We decided to investigate the CPIM format and as a fallback solution write our own intermediate format if CPIM turned out to be too troublesome. Using IETF drafts suggesting SIP, CPIM and XMPP interworking we made tables of conversion rules and allowed characters. Using those it was straightforward to implement the CPIM format in our transformer classes.

**Deferred Messages**

Deferred message in Fuatara's gateway is preferred to be called as SIP deferred message. Messages sent from XMPP client to SIP client through our gateway might encounter situation that SIP user is not online. In this case, these messages should be temporarily stored in gateway's database as deferred messages and will be resent next time when the SIP user is online. The reason why the gateway only handles deferred messages to SIP not to XMPP is that OpenSER, the SIP Server, will not handle the deferred message itself, but OpenFire, the Jabber Server, will.
A MySQL database is set for deferred messages. An incoming message from XMPP to SIP is always stored in a heap temporarily. It will be removed from the heap if a 200 OK notification is received indicates that the SIP Client has received the message. Otherwise, the message will be move to the database as a deferred message. Every time a SIP user sends an online notification, gateway will check the database to see if there is any deferred message for the user. If yes, deferred messages will be retrieved and send to the user.

Development History
The choice was made to develop our own gateway after surveying existing open source SIP/XMPP gateways. The only real candidate for this seemed to be OpenSER which includes some gateway functionality. However we were unsure how well it would work and its source code was not very documented nor commented. So if we were going to make any changes to it this seemed to be a cumbersome task.

The original project proposal suggested developing a SIP servlet using Java EE. A suitable open source sip container was found in Sailfin and we started developing our gateway for it in Java SE. The decision not to use Java EE was based on the fact that the group’s total experience of the enterprise version was very small. The go ahead decision for using Java was made after a discussion about various languages and their suitability. For the gateway we had no explicit performance demands so Java was chosen because it was determined that the transition to doing actual developing work would be easier since a lot of the sample open source code, in form of tutorials, was in java. Also it was thought that development in Java would be faster and a good fit for the Java Jabber server component already chosen.

The decision to abandon Sailfin in favor of Jain Sip was not based on any comparative study between those two. Instead it was the result of Jain Sip being more practical for us to use. The parts of the team responsible for the gateway had a startup phase of three weeks were knowledge were gathered, open source downloaded and development environments set up. During that phase a lot of sample code was downloaded, some of them using Jain Sip. The team faced two tasks at the same time, first to learn the structure and details of SIP and XMPP messaging and second how to actually write code for it. The team soon run into some problems, we were unable to get all Sailfin tutorial servlet code collected to work on Netbeans. With help from Ericsson we get a working development environment with Eclipse instead. The next major problem was that we were still learning sip servlet coding and we experienced sample code running on one version of Sailfin but not on another. We had to limited knowledge to determine if the code was wrong or the problem was on the server. In addition to that the startup time for Sailfin on our equipment used was several minutes, which meant we spent a lot of time waiting while trying out servlet code, our alterations to it and different combinations of Sailfin versions.

During the gathering of sip servlet tutorials and sample SIP applications we had gathered some using Jain Sip. Those sample applications we had no major problems with and we could get them to run and make changes to them easily. A first framework for the gateway was done with Jain Sip and used from that point.

By the end of October we had a working gateway and started to make the final integration with the Jabber Server. This integration resulted in changes to the libraries we had used for Jabber
communications. It was decided to run the gateway as an external Jabber component. The SMACK library we had been using so far could not be used for such connection so instead we searched and found the JabberWookie library. After the server integration had been completed we added the ability to handle delayed messages, so a user can receive old messages when getting online.

Once the message and presence functions were working we had time over to features desired but not required. This included gateway support for other protocols such as MSN. Work was started on modifying the gateway. The present gateway is not responsible for those conversions since a quicker way was found by using Open Fire to handle this.

Testing
As mentioned, the gateway development has not been geared towards performance or robustness as the main goal. Though we have taken some measures to prevent it from crashing when receiving wrongly formatted xml stanzas and detecting if the connection to the server is lost, there has been no real testing of its performance and robustness. Instead the testing has been conducted during the continuous integration process. This means in practice our testing and development has been interleaved and focus has been on getting the presence and messaging to work. The design goal of easy extension to new protocols also remains untried, this because the handling of other protocols become part of the Jabber server.

Fuatara Client
The Fuatara mobile client is based on MGTalk which is a Jabber client for the J2ME MIDP 2.0 platform. It was chosen after a survey of existing opens source clients. We had no problems in getting it to run on our target phones and the code was easy to read so it seemed a good choice. This choice was made very early in the project so we have had the opportunity to spend some time with it. We have modified the original program so we now have a client with the following functionality:

- Support for MSN, ICQ, AIM and SIP messaging
- Blog posting with images, audio and video clips
- Reading of notifications subscribed through the Fuatara web site.

The client communicates with standard XMPP messages so it should work with any XMPP client.

Development History
The original MGtalk client has gone through some major changes. The original code framework was retained during the development of multiple protocol messaging and blogging capabilities. When we had those functionalities the decision was made to make changes to the code in order to have a better code structure and improve the ability to run on other phones than our target phones.

To improve support for a wider range of phones we briefly used the Polish framework. This line of development was abandoned after we run into problems using the Polish framework for the multimedia capture. Using this framework we were not able to get the media capture to work properly even on our target phones.

The other major change introduced was the adoption of the Beep framework. Beep is a wireless telecommunication framework for the Jabber protocol. By plugging in an existing Jabber framework we got a better modularization of the server communication code. The remaining code was also
completely refactored to get a better code structure. The clients classes was structured into a group of classes with the suffix Manager with different responsibilities, e.g. ChatManager which is responsible for messages, including showing the income message and sending message out, and transform text into graphical smiles.

Working with Eclipse and the SDK from Sony Ericsson turned out to be rather problem free. We never experienced any hard times with the emulator and the ability to do online debugging on an actual phone also worked very well, and was extremely useful.

**Development Issues**

Problems were encountered in the area of media capture. A common layout for audio, photo and video capture was not possible since the camera preview could not be displayed in a Java Form originally wanted. Audio capture uses a Form but the camera preview is now drawn directly on a canvas. The problem encountered was that there was a difference between the orientation of the picture when the camera operates in preview and capture mode. When a picture is taken or a recording is started the display gets rotated by 90 degrees. The Java Form could not be rotated to counter this. We found out that this problem was reported on the Sony Ericsson developer forum as well as on Nokia forums. For certain mobile models this problem exists. Various solutions suggested there did not work on our target phones. Instead we opted for a workaround of holding the camera in landscape view and when recording video skip the preview mode and skip the preview mode. Instead the camera continuously records but discards it until the user starts the actual recording. When doing media captures there were also problems of out of memory situation on the phone. This was solved by doing manual garbage collecting and limits of media sizes.

We noticed some notifications were not showing their content properly. However, we worked this around so it is still possible to go to the URL of the notification and read the update directly. Another sacrifice is that we use plain text for authentication since we could not get SSL to work with the original MGTalk.

**Known Unresolved Issues**

- Support for other phones than our target phones is still very unsure. From the limited testing done we can conclude that there is no guarantee for the application to work properly.
- The Beep library has problems with international characters. According to the author this should have been fixed in version 1.0 which we are using but we are still unable to receive Swedish characters on the mobile. By explicitly telling to use UTF-8 encoding we have fixed the sending of Swedish characters.
- Testing on 3G network revealed that the battery will not last very long. Here we tried to increase the ping time interval but were unable to find any settings for changing it.
- As part of the refactoring process the graphical interface was also redone. However we have done no real user testing of the interface to see how well it works and how it is appreciated.
Open Fire components

OpenFire Server

Openfire is a Jabber server dual-licensed under GPL and commercially. Since the code of this project is going to be licensed under GPL, we may use it under the rules of the GPL license. It uses XMPP as the protocol for instant messaging. The easy setup (web-based), the performance and its good security reputation lead us to choose it among other good Jabber servers. We also take into account that was written in Java, and since every other component of the project was going to be written in Java, it made sense to use it, in case we had to adapt or write a plug-in ourselves.

IM Gateway Plug-in

We use this plug-in offered by Ignite Real-time, in order to provide connectivity with the other public IM networks, such as MSN, AIM or ICQ. We have enabled the three of them, since those are the ones that we support on the mobile client, but this can be extended to support other IM networks in the future. Users are able to communicate to their buddies in other networks using their jabber account, as long as they register themselves with the specific protocol providing their username and password.

Open SER

The SIP server acts as the endpoint for the desktop clients and allows them to send messages to other SIP clients. The SIP server is also capable of forwarding messages to other users not using the SIP protocol. Currently we are not using a server made by ourselves.

The server used is OpenSER and we are using version 1.1.1. We run the server right out of the box. First we run it with a small change to the basic setup. OpenSER uses a script for such functions as routing messages. We used a scheme for users contact address that will tell the server if the address is a SIP or Jabber user. Jabber users will be on our gateway address so the server looked for this and if the address is found then forwards the message to the gateway. Our present Openser and gateway does not need any changes to the scrip. A Fuatara contact in the sip client will look like <user>’Fuatara.com@<gateway ip>, so the message will go directly to the gateways address.

The server we use is noted for its high performance, but that was not the main reason behind putting it into use. The original plans of developing a SIP servlet for the Sailfin container proved more troublesome than expected and instead we opted for an existing server implementation that suits our needs well and are also relatively easy to work with. Without too much trouble we were able to get Open SER up and running and it needed only small changes to its configuration scrip.

Problems encountered

First we choose coding our own Wordpress plug-in to post image, audio and video on blog. Then we found Wordpress uses an xml-rpc interface which makes user can post blogs using many weblog clients. Because it has most of functions we need, so we gave up developing our own plug-in turn to use xml-rpc. However, it runs well randomly just on some certain media files. After several times of debugging, we found we need to edit source files of Wordpress to make it run well on all media files, which means if we use xml-rpc then our solution would be blog dependent. We tried some way to get it through, however it seems impossible except edit the xml-rpc source code. So, at last, we
found Atom, an up-and-coming format for editing, syndicating, and archiving weblogs and other episodic web sites, furthermore it is implemented and implementable by everybody, so most of weblogs support it. We used atom instead and the problem solved quickly.

We use a Wordpress plug-in named “WordPress Mobile Edition” to show a mobile view of posts. Our design is to give a link to media file in the post, and what we need to support is audio file .amr and video file .3gp. It runs fine on posts with just .3pg files, but it just gives a text format of .amr files instead of playing it. Use Wireshark (a network protocol analyzer) to research what content type the server send to mobile browser. It shows Wordpress just use text plain to open the audio file. So we need to use .htaccess file which provides a way to make configuration changes on a per-directory basis, to add .amr to be handled as an audio file. This also led to edit httpd.conf file, the main configuration file of Apache, to enable .htaccess. However it just plays the .amr file along as background audio file, so when there are many audio files in one post, it would be noisy and slow for there are too much data to download. Therefore we create a simple php file for just using the default player to play .amr files and give links to these files on the page of post.

Project methodology and organization

Scrum Overview

Scrum is an agile development method that focuses on the team’s independence and self support. It is an iterative method which produces a “new” extended product at the end of each iteration. Scrum is a method that allows a lot of freedom in the way of how you integrate it. But there are a few guidelines that we have followed. First there shall be a Product Owner (PO) (which in our case was what traditionally is called the project leader). His job is to be the contact person for the team. The team should be able to work focused and undisturbed during a Sprint (iteration). It is the PO’s job to have a backlog with features in a prioritized order of the system. The team decides how many features they think they can implement from the top of the backlog during one sprint.

Sprint

A Sprint is the time the team decides that they need to be able to implement the next features from the backlog. The PO will also have an estimate of how fast features are implemented (see “Planning meeting” below) and can somewhat make up a time plan of the project when different features will be implemented. This is easy because the scrum masters daily update a “Burndown” chart that shows how much work has been done and how much work that is left during the sprint.

There are not allowed to be any changes of the sprint backlog during a sprint, so if the customer wants to change anything they have to talk to the PO who changes the backlog which will be implemented during the next sprint. This is so that the team will work focused towards a predetermined goal during every sprint. If features are removed or added a new time plan will easily be made by the PO and this way the costumer will have an up to date time plan of the project easily at hand.
Planning meeting

At the start of every sprint the team meets up and together decides what goal the next sprint should have. The team also decides what features to implement next and discusses the design of those. All features in the backlog are decided and agreed upon by the PO and customer. The customer may at any time come with suggestions and changes. Any changes that come up during a sprint are not implemented by the team until the next sprint.

Furthermore the team has to decide what approximate points they think every feature will take. This is done by playing cards, e.g. every team member gets a suite of cards with numbers. Then everyone decides a number and puts it on the table in front of him face down. When all cards are on the table they are turned. Discussions arise if there is a larger discrepancy and the team members argue for their number chosen. This way we see if anybody has misunderstood the feature or even better if someone has a great knowledge or former experience with this kind of feature. The numbers are compared with former similar features and in this way the approximation will be more and more stable during time.

Daily Standup

Normally everyone in the team attends a daily standup meeting where everyone in the team should answer the questions

- What did you do yesterday?
- What impediments do/did you have?
- What are you going to do today?

This allows the team to know what everyone else is doing and quickly take action if anything needs to be fixed e.g. design decision or bug fixing. Scrum exists of many meetings but they should be kept small. Normally a daily standup is no more than 15 minutes. It is in the scrum methodology’s belief that direct face-to-face communication is the key to fast development and successful software. This is also one of the main tasks for the scrum masters to see to that the team follows.

End of sprint review

At the end of every sprint the team meet again and have an end of sprint review where everyone say what they think were good during this sprint and what they want to change. This way the team improves and all members have the possibility to make changes of how they work as a group. Typically all teams meet up with the PO and the customer for a demonstration of the latest product (latest features implemented). This way the costumer will have continuous feedback of how the project evolves and the teams will easily receive feedback from the costumer.

Scrum – how we used it

Why scrum

Scrum was introduced to some of the students during a class in software engineering at Uppsala University. Since none in the project had former experience with working with a project methodology we chose scrum because it’s an iterative developing method which seemed to suite us well because we frequently had an extended working version of the system which we could demonstrate and get feedback on from our customer (Ericsson Research).
Implementation

We started out as one large scrum team with 10 students and one scrum master. Normal a scrum team exists of 3 to 8 persons. We read through the agile manifesto and “Scrum and Xp from the Trenches” by Henrik Kniberg and started to implement our version. Daily meetings, sprints, backlog with features to implement and a scrum board.

Figure 8: Scrum board at the end of Sprint N-4

Organizational changes

After having the possibility to attend a seminar about scrum with Henrik Kniberg and Jeff Sutherland, one of the “co authors” of scrum, we decided to restructure a bit. Our scrum master at that time discussed with Jeff Sutherland and he suggested that we should split up our team into smaller groups to improve productivity. It should be possible to increase your productivity about 2 times according to Jeff Sutherland who had some figures from a company that had done some studies and tried this. He said that it later has been more common with smaller scrum teams with 3-4 persons rather than 7-8. With this in mind we decided to split our team into two groups of 5 students each.

Two new scrum masters were assigned from each team and the former scrum master became our PO (process owner). Since the workload on this position wasn’t comparable to the other development positions the PO also became one of the team members in one of the teams, resulting in that one of the team existed of 1 PO, 1 Scrum master and 3 other students. Not the optimal structure with a PO within the scrum team but it has worked well for us and the PO could attend all meetings since we chose to have the two team sprints interleaved.
The main purpose for the scrum masters was to glue the group together and make the team work in the same direction. One team focused on SIP/XMPP IM communications as well as on the mobile client. The other team developed the server structure for notifications as well as the web managing system. The two scrum masters together with the PO answered for the integration between the two teams so that there were no design issues or other team collisions.

**Retrospective**

The common feeling is that scrum has really been a help within this project. When we look back at other project reports from earlier years we can see that they still hadn’t start coding in early November where we started of only a few weeks after the project launch. We had a basic version up very early and we think that it has been a great value to us since the project existed of a lot of different parts that were to be assembled into one system. It has been much easier to develop the system correct from the start since we have been able to give and receive feedback both from and to our customer throughout the entire project.

The daily standup meeting has also been a grand help in the sense that impediments and problems have easily been detected and a quick solution has been decided upon by the team. One example to this our standup meetings started with one of the team members picking up a token (a blue whiteboard pen in our case) and presented his part, followed by handing over the token to any person of his choice. The problem was that no one wanted to start talking. The suggestion from one team member were that anyone could pick up the token at the start of the meeting and directly hand it over to another member who had to start talking. Now it suddenly became very interesting to get the token first and give it away. No more silence in the beginning of the meetings.

The reason for not having the scrum master staring to talk every time is because the team should work as a team and the scrum master is not the boss over the other in that sense. In addition the team members should not talk and explain to the scrum master; rather, they should be talking to the whole team.

**Continuous integration and testing**

**SIP**

The team had no previously experience of the SIP/SIMPLE protocol. Since we had to use it the highest priorities went into understanding it and find software for it. Since this protocol was to be used on one of the IM clients we can only say something about its functions for messaging and presence. The protocol as such was first thought to be a major bottleneck four our initial progress, but as it turned out its usage and conversion into XMPP was possible without much trouble.

The real, and not foreseen, problem was in the area of early releases of SIP software. This is especially true for SipCommunicator which we centered our development around. Although new versions of SipCommunicator were released we come to know it as very unreliable. The new releases never solved problems but instead created the additional burden of keeping track of what worked in what version. The lesson learned from this is that the disclaimer about its functionality show when starting up is very serious.
The other major SIP open source component we tried out was SailFin. Here we run into the same problems as with SipCommunicator. It seemed that SailFin did not like to be restarted a lot on our equipment. We had a lot of problems just setting the development environment up and figure out which version of SailFin was the best to use.

As the team were new to programming for SIP it was important for us to early gain knowledge in this area. This proved to be no problem since we were able to quickly gather a wide variety of sample code and tutorials. For someone just wanting to take their first steps in SIP development this was sufficient. But the team also found out once you have mastered the basics the documentation and tutorials for more advanced things were very scarce. Our impression is that a few basic tutorials have been spread and is to find at various places in more or less the same version. What we really missed was sufficient detailed instructions on the SailFin project page to get it up and running the sample code, both in the provided NetBeans project and with Eclipse. We were unable to run all the sample code provided for SailFin.

**XMPP**

The teams experience with the XMPP parts are generally a good one. Early on in the project we decided on MGTalk and OpenFire. Those two components, especially OpenFire, we view as stable compared. Though MGTalk was very low in its version number it has caused no stability problems for us. From our limited experience we have come to view XMPP clients less experimental than their SIP counterparts.

The team has had no major problems in finding and using XMPP documentation. The fact that our selected Jabber server worked very well meant that we could concentrate the XMPP work on the gateway. Here we could not find a true answer to how the conversion to SIP should be made. We relied on IETF drafts for guidance and were by means of those able to get the XMPP part working with the SIP part.

**Conclusion and future work**

*Experience using Open Source*

*The IM teams experience*

The project was to be based on open source components. That meant we, in theory, had the opportunity to speed up development by using existing code. However, the use of open source introduced other problems which were not fully expected from the start. First, once we had a first overview of what our system would look like the search was initiated for suitable open source components. This search of course took some time but the main issue seen is on what grounds should we use a particular piece of software? The performance aspect of our system part was of less importance. But it easy to understand that if there had been performance demands on our components, then test methods must be set up to verify the performance of a component under evaluation. Our general impressions are that performance descriptions are often missing in open source, at least for everything we looked into. Our conclusion is that our task would have been much more difficult if we had to choose open source based on performance criteria. The second main
problem was that the open source components we looked into were not fully developed. That meant they come with very little, if any, guarantee to work in our particular setting. When selecting open software it was very useful to also have a look of the things working not so well. This includes searching forums for problems, looking at their reported bugs and searching on the internet for people’s opinions of the software we were evaluating.

The IM team had no demands like we must use an open source component for a server or the gateway. We always had the choice of developing our own code. For the gateway module we did so. Here we really could not find any suitable existing open source equivalent. This did not mean we did not benefit from open source when making this part. Because we now were not confided to a single program we could benefit from pieces here and there when setting up our first version.

The quality of the documentation for the open source we used could have been a real problem for us. All documentation is geared towards describing the product in a very shallow way, often it is just some Javadoc. In practice this has been no problems for us, but the provided documentation has been of very little use when it comes to understand how different parts in the program are connected.
Appendix A: Installation Instructions

Fuatara is developed with, and only tested for, the following environments:

- Apache 2.2.3
- Java 1.5
- MySQL 5.0.38
- PHP 5.2.1

Please ensure you have at least these versions of the software components required, or you risk the different components not working.

A1: Installation Instructions for Openfire

Installation

- Get the tar.gz Linux package of Openfire (available at [http://www.igniterealtime.org/downloads/index.jsp](http://www.igniterealtime.org/downloads/index.jsp)).
- Extract the package with "tar -xzvf openfire_3_4_1.tar.gz"
- Move to /opt with "mv openfire /opt"
- Use the bin/openfire script in your Openfire installation to start the server: "#/openfire start"

Setup Database

- Create the database for Openfire using the openfire_mysql.sql schema file: “cat openfire_mysql.sql | MySQL”
- Launch Openfire and use a web browser to connect to the admin console. The default port for the web-based admin console is 9090. If you are on the same machine as Openfire, use the following URL: [http://127.0.0.1:9090](http://127.0.0.1:9090).
- In the Openfire web-based setup tool, use the following values, where [YOUR_HOST] and [DATABASE_NAME] are the actual values for you server (in many cases localhost is a suitable value for [YOUR_HOST] when your database is running on the same server as your webserver): “driver: com.mysql.jdbc.Driver”, “server: jdbc:mysql://[YOUR_HOST]/[DATABASE_NAME]”

Setup Init Service

- If you would like to install Openfire as a service, one script is provided in the bin/extra directory: openfired.
- Before running this script make sure $OPENFIRE_HOME/bin/openfire is executable by the user you want to run openfire: “chmod +x $OPENFIRE_HOME/bin/openfire”
- Copy the script into /etc/init.d.
- Link the script to your default runlevel directory: “sudo update-rc.d openfired defaults”

IM Gateway Plugin

- Download directly inside the administration console. It will automatically be installed.
The plugin is configured via the "Gateways" sidebar item located in the Openfire Admin Console. You can enable individual transports via the "Settings" sidebar item, and add new registrations/view existing registrations from the "Registrations" sidebar item.

**Essential Configuration**

- There are several steps to config to ensure the openfire is work with our project.
- 1. Log in openfire as an admin
- Enable external component with port:5275 and secret:cct.
- Either set Allowed to Connect to Anyone or set a Whitelist of the subdomain allowed
- (Note that subdomain and secret should be the same as in the gateway configuration file : xmppservers.xml. See Gateway deployment instruction).

**A2: Installation Instructions for Gateway**

**Extract Package**

- If you want to run it under windows, extract it and move the gateway folder under any directory
- Under Linux system, use “unzip gateway.zip”, ”mv gateway [anydirectory]”

**Database**

- Create a database, simply write “CREATE DATABASE [databasename]” at the MySQL prompt and change the [databasename] to anyname you like to call the database, like "gateway_db"
- Create a table in the database, write the following to the MySQL prompt and change the [databasename]: “CREATE TABLE `[databasename]`.`delay_message` (`id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, `message_from` VARCHAR(45) NOT NULL, `message_to` VARCHAR(45) NOT NULL, `via` VARCHAR(50) NOT NULL, `content_type` VARCHAR(45) NOT NULL, `body` VARCHAR(1000) NOT NULL, `content_encoding` VARCHAR(45) NOT NULL, `content_length` VARCHAR(45) NOT NULL, `call_id` VARCHAR(200) NOT NULL, PRIMARY KEY (`id`)) ENGINE = InnoDB;”
- If the client reponses with Query OK, you have done with the database setting

**Configuration**

Under installation directory gateway/ there are several configuration files

1. dbconfig.xml
2. protocols.xml
3. transformers.xml
4. sipconfig.xml
5. xmppservers.xml

- **dbconfig.xml**: configuration about MySQL database
  - `<connection_url>` is the connection string of MySQL database
• [host] is host address of MySQL deployed
• [port] is port number of MySQL
• [databasename] is the name of database you set
• An example of <connection_url> is like this:
  jdbc:mysql://localhost:3306/gatewaydb
• <username> is the username of your MySQL database, like root
• <password> is the password of your MySQL account, like 1234

```xml
<?xml version="1.0" encoding="UTF-8"?>
<connection>
  <connection_url>
    jdbc:mysql://[host]:[port]/[databasename]
  </connection_url>
  <username>username</username>
  <password>password</password>
</connection>
```

• **protocols.xml**: configuration file to set protocol user want to use in the gateway *(If you use XMPP-SIP gateway only, leave it default)*
  • <protocol> is a single protocol
    • <protocol-name> is the name of the protocol, like "SIP"
    • <protocol-class> is the java class name of the implemented protocol, like "se.uu.it.cct.gateway.SipLayer"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<protocols>
  <protocol>
    <protocol-name>protocol name 1</protocol-name>
    <protocol-class>class name 1</protocol-class>
  </protocol>
  <protocol>
    <protocol-name>protocol name 2</protocol-name>
    <protocol-class>class name 2</protocol-class>
  </protocol>
</protocols>
```

• **transformers.xml**: configuration files to set transformers *(If you only use XMPP-SIP gateway, change the gateway domain)*
  • <gateway-domain> is the address of the machine where the gateway set to, like "130.238.15.193"
  • <transformer> is a single transformer will be used in transform between different protocols
- `<transformer-name>` is the name of the transformer, like "SIPtoCPIM"
- `<transformer-class>` is the java class name of the transformer, like "se.uu.it.cct.gateway.util.SIPtoCPIMTransformer"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<transformers>
  <gateway-domain>gateway domain</gateway-domain>
  <transformer>
    <transformer-name>name 1</transformer-name>
    <transformer-class>class name 1</transformer-class>
  </transformer>
  <transformer>
    <transformer-name>name 2</transformer-name>
    <transformer-class>class name 2</transformer-class>
  </transformer>
  <transformer>
    <transformer-name>name 3</transformer-name>
    <transformer-class>class name 3</transformer-class>
  </transformer>
  <transformer>
    <transformer-name>name N</transformer-name>
    <transformer-class>class name N</transformer-class>
  </transformer>
</transformers>
```

- `sipconfig.xml`: configuration file of sip protocol layer
  - `<stack-name>` is the name of sip stack, set as "CctGateway" if you use the siplayer we provide
  - `<gateway-ip>` is the address of the machine the gateway deployed on, like "130.238.15.193"
  - `<gateway-port>` is the sip port gateway use, like "5060"
  - `<outbound-proxy>` is the sip server the gate connect to, the server can be openser or others, the setting is like "130.238.15.239:5060"
  - `<trace-level>` leave it as default
  - `<server-log>` leave it as default
  - `<debug-log>` leave it as default

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sip>
  <stack-name>CctGateway</stack-name>
  <gateway-ip>130.238.15.193</gateway-ip>
  <gateway-port>port</gateway-port>
  <outbound-proxy>proxy</outbound-proxy>
  <trace-level>32</trace-level>
  <server-log>gateway.log</server-log>
  <debug-log>gateway_debug.log</debug-log>
</sip>
```
xmppservers.xml : XmppServers setting configuration file, since xmpplayer can connect to several Jabber server

- <server> is one XMPP server the gateway connect to
  - <server-name> is the url address of the XMPP server, like "parthenon.d2dx.com"
  - <server-port> is the port of the XMPP server used to establish a component-server connection with our gateway, like "5275"
  - <server-secret> is the secret XMPP server set when establish a component-server connection with our gateway, like "cct"
  - <subdomain> is the sub domain used, as address of the gateway, like "sip"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<servers>
  <server>
    <server-name>server_url 1</server-name>
    <server-port>port 1</server-port>
    <server-secret>secret 1</server-secret>
    <subdomain>subdomain 1</subdomain>
  </server>
  <server>
    <server-name>server_url N</server-name>
    <server-port>port N</server-port>
    <server-secret>secret N</server-secret>
    <subdomain>subdomain N</subdomain>
  </server>
</servers>
```

Run

- Go to the directory gateway installed and run "java -jar Gateway-JainSip.jar"

A3: Installation Instructions for Website

- Download the Fuatara website package from http://Fuatara.org/downloads/Fuatara_web_1_1.zip
- Unzip it to the destination directory.
- Create a database in the target MySQL server.
- Set up the Database tables by executing the db_schema.sql file.

To get the website to work correctly now, two files need to be edited.

General Configuration

This sets up several variables that allow the website to do accurate links and store session information. Edit the following file:

/system/application/config/config.php

and set the following values:

- $config['base_url'] = "/";
• Set this to the base URL of the Fuatara website, if it is different from the root. Remember the trailing slash!
  • $config['cookie_prefix'] = "";
  • Set this to a string using only lowercase latin characters (a-z) that represents your domain and is relatively unique.
  • $config['cookie_domain'] = "";
  • To increase security, set this to the domain of the website for increased security

**Database Configuration**

Now the database settings need to be added to the configuration. Edit

*/system/application/config/database.php*

and set the following values:

**Default**

This references the database to be used by the website and notifications servers.

  • $db['default']['hostname'] = "localhost";
    • The MySQL hostname.
  • $db['default']['username'] = "";
    • The MySQL username.
  • $db['default']['password'] = "";
    • The MySQL password

**Login**

Set the same values as above, but this time to the login information used for the Openfire installation (see above).

**A4: Installation Instructions for Notification Servers**

The following instructions refer to version 0.5 of NotifServers (r996).

  • Download the NotifServers package from
  • Unzip to any directory.

The NotifServers package comes with all libraries required for successful execution. All that needs to be done is some configuration.

**Configuration**

All servers use the standard CCT Config file, an example of which can be located at


For a single-computer setup, all that needs to be done is to set up the database section (this must match the settings entered in the database) and the email sending section. Please note that using local sendmail is unsupported; we require SMTP service.
Once configured, you can optionally place the config file in the default search path (/etc/cct_config.txt).

**Launch**

To launch the servers, simply run

```
./start.sh ./cct_config.txt
```

Or if you put the `cct_config.txt` in `/etc`, you can do simply

```
./start.sh
```
Appendix B: User Guides

B1: Mobile Fuatara Client

Installation

- Load the jar file into the mobile phone using bluetooth, an usb cable or infrared.

Instructions

Login

Open the Fuatara client. It should be under the applications or games folder. Fill in your username and password. Optionally check the auto login option if you want to be automatically logged next time you open the client.

---

![Login Window](image.png)

Figure 9: Login Window
Third Party Protocol Settings

To change your server information, choose Settings from the main menu. Then choose Server. You can choose the domain and port you wish to use, and say whether or not you wish the application to auto-connect upon startup.
Alternatively, you can choose Server when you are at the login screen to be given the same options.

**Chat**

**Send Message**

Scroll down to the contact on your list that you wish to send a message to. Choose Chat from the menu. In the chat window, scroll down to the message field. Write your message. Choose OK from the menu, then choose Send from the menu to send the message. It will be displayed on the screen along with the messages sent to you by that contact. Choose Back from menu to return to the main screen.

**Forward Message**

Press the forward command of the menu. Choose one or several users in the new screen and press forward to send the message to those users.
Open URL

You can open a url embedded in a message choosing the Go Url option in the submenu.
Change Status

From the main menu, choose Status. From the status screen, you can choose which custom message will be shown to your contacts, set your status to Online, Away or Do not Disturb. Choose OK to accept your new settings, or choose Back to leave the status screen while keeping your original settings.

Log to Other Networks

From the main menu, choose Status. Choose whether to log in to MSN, ICQ and AIM with your id and password for that respective protocol.

Add a New Buddy

To add a new contact, choose Add from the main menu. Enter which protocol the contact is using (Jabber, MSN, AIM or ICQ). Enter the contact’s id, according to the protocol used. It is also possible to name a SIP contact on the Fuatara side by using <user>%<Fuatara sip server ip>@sip.<OpenFire domain>. For example the contact’s name is joe, OpenSer is running at 130.238.15.169, and OpenFire is at Fuatara.com then your contact’s address will look like this: joe%130.238.15.169@sip.Fuatara.com.
You may be asked whether the other user may add you to their contact list. Answer Yes or No.

![Figure 19: Add new buddy](image)

**Delete User**

Scroll to the contact you want to delete in the contact list. Choose Delete from the menu. Answer Yes when asked for confirmation that you want to delete this contact.

![Figure 21: Delete buddy](image)
Notifications

Read Notification

From your list of contacts, choose the Notifier. You will see a list of notifications. Scroll to the one you want to see and choose Read from the menu. You will be shown a brief description of the post you have been notified of. To return to the notifier screen and choose a new notification to read, choose Back from the menu. From the notifier screen, choose Back to return to the main screen.
Load Image

When you press one of the notifications on the list, a new screen will appear showing the content of that notification. It's up to the user to load the image from the website (in case is embedded in the notification). To load it, you must click the option Load Image on the submenu.
Open URL

To start the web browser and go to a post you have been notified of, choose Go to URL from the menu when looking at a specific notification.

Forward Notification

When viewing a notification, choose Forward To from the menu. On the list that appears, mark the contacts you want to forward it to. Choose Forward from menu to forward the notification and return to viewing the notification. Choose Cancel from menu to return to viewing the notification without having forwarded it.

![Figure 27: Forward Notification](image)

Delete Notification

On the notification screen, scroll to a notification you want to delete. Choose Delete from the menu to delete. Alternatively, choose Delete All to delete all currently stored notification.

![Figure 28: Delete Notification](image)
Blogging

Create Post

While Fuatara is showing your contact list it is possible to also post to your selected blog at Fuatara.com

- Select Menu and choose Blog in the menu that appears
- The post entry form is then shown. Here you can enter a text message. In the sample illustration the user wants to post a "Hello!" message to her blog.

![Figure 29: Post to your blog](image)

Browse File

To do this select Menu and choose FileBrowser.
Add Photo

- It is possible to take a photo and post it to your blog.

To do this select Menu and choose Add Photo. You will be taken to a photo preview screen. Choose Large or Small from the menu to set the resolution for taken pictures to high or low. Click the button on the side of the mobile (or alternatively the FIRE button) to take a picture. Choose Send to put the new image file in the blog post. At any time, choose Exit from the menu to leave picture-taking mode without saving a picture.

Add Audio

It is possible to add recorded sound to your blog post.

To do this select Menu and choose Add Audio
Add Video

It is possible to record a video and post it to your blog.

To do this select Menu and choose Add Video. You will be taken to a video preview screen. Click the button on the side of the mobile (or alternatively the FIRE button) to start the recording. Click again to stop the recording. Choose Send from the menu to put the new video file in the blog post. Choose Exit from the menu to leave video-capture mode without saving a video.

B2: Using The Fuatara Web Site

1 Registering on the Fuatara site

The webpage for registering a new account is located at: www.Fuatara.com.

In order to register, select Register in the Main menu on the site. Username and Password is required, Email is not required, but you won't get e-mail notifications if not specified. It can be specified later in My profile, available in the main menu when logged in. Registering will
automatically create a jabber account with the same username and password on the server that you can use with a jabber client (more on that later).

2 Logging In
Log in by selecting Login in the Main menu. Fill in the required fields.

3 Subscribing to Blogs and publishing new Blogs
To subscribe to a blog from the Blogs menu select either search or most popular. Click on subscribe for the blog of your choice. Make sure the kinds of notifications you want are checked (and those you don't want should not be checked). You may add filters, each filter is for those types of notification only, in order to get a filter you must specify wither it is for title, contents or both. If exclusive filter is checked then the word given must NOT appear in order for the notification to be received.

To publish a blog select add blog. That blog must have our plug-in to be able to send notifications. The Blog name and password given here must be the same as defined in the options of the plug-in. The remote posting information is only required in order to quickly post to your blog from our mobile application, and it will only go to whatever blog is selected as default (done from My Published). That information is username and password for a user of the blog that it can post as.

4 Rss Feeds
Subscribing to a rss-feed is in many ways similar to subscribing to a blog. Either listing feeds using the Search or Most Popular menu. However rss-feeds are not published, so any user may decide to add any rss-feed (by the use of Add Rss feed) though remember that the URL given there is the URL of the actual feed.

5 Buddies
List buddies will list all the buddies you have to your jabber account. Add buddy is used to add a buddy obviously, nickname is simply a nickname you wish to refer to that buddy as while id is the buddies id in the form of username@server (unless the buddy uses another protocol then jabber such as SIP, in which case it is username@gateway). Protocol is what protocol the buddy uses, jabber is default.

6 Log Out
To log out simply select Log Out in the Main menu.

7 Notifications
Notifications are received as soon as a Blog page or a RSS feed which you have subscribed to is updated.

7.1 Instant Message Instant Message notifications are received in you’re logged in IM client using the same account as which you use logging in to the Fuatara site. A few sentences of the Blog entry or Rss feed will be shown as well as the title and an URL (link) to the actual site.

7.2 Widget The Notifications received are presented to the user in the form of a "bubble" appearing on the screen. This appearance is generated by OS-native applications (see below) which present
users with an appealing interface that they can configure. When the notification appears, the title will be included in boldface, and an excerpt of the notification text presented to the user, giving him/her an overview of the information presented. The notifications can be left-clicked on. Doing so will lead the user to the webpage that caused the notification to happen. This allows the user to quickly access important data. The notifications can also be right-clicked for them to be instantly dismissed, without any action taken. This allows them to be relatively unobtrusive to a user’s workflow.

The Fuatara Client is dependent on Snarl for Windows and Growl for Mac OS X (Growl will be installed during the client installation. These are system-wide notification agents that allow users to manage notifications from several sources at the same time. The desktop client (widget) itself can be downloaded for Mac and for Windows.

7.3 Email notifications are sent to the email address which you entered during the registration on the site. The email address can be edited under the Main/My profile menu.

8 Mobile Client
The mobile client is called Fuatara and can be downloaded from our website Fuatara.com. With the mobile client you can send and receive instant messages from your friends, receive notifications if you have subscribed to any blog or rss feed and you can post on your default blog page (choose default blog page under My published blogs).

9 Wordpress Plugins
In order for a blog to send notifications it needs our Fuatara Notifications plugin for Wordpres blogs. It is activated as any other plugin, do note that in options it will add options for Fuatara Notify. Weblog Name and Weblog password there MUST be the same as is given for the blog on the Fuatara site for it to work. Fuatara server is simply to which server it should send it to (and should by default point to the right one).

B2: Adding a Fuatara contact in SipCommunicator

Login
Create New Account
Using SipCommunicator it is possible to talk to other Fuatara users which are online on their mobile. First you choose to create a new account. Fill in your user name, password and address of SIP server (for example, OpenSER server).
**Add Buddy**

**Create Group**

Before adding a new contact to your buddy list is necessary to create a new group. Use the create group wizard to create a new group.

**Add Fuatara Buddy**

To add someone not using SipCommunicator you will have to fill in your contact address following the below rules:

```
<user-name>'<jabber server domain>@<gateway ip>
```

For example, you have a account in Jabber server: a_user@Fuatara.com, a gateway with address 130.238.15.193 and Open Fire running at Fuatara.com, the contact address should be like this: a_user@Fuatara.com@130.238.15.193.
Appendix A: Installation Instructions

Fuatara is developed with, and only tested for, the following environments:

- Apache 2.2.3
- Java 1.5
- MySQL 5.0.38
- PHP 5.2.1

Please ensure you have at least these versions of the software components required, or you risk the different components not working.

A1: Installation Instructions for Openfire

Installation

- Extract the package with "tar -xzvf openfire_3_4_1.tar.gz"
- Move to /opt with "mv openfire /opt"
- Use the bin/openfire script in your Openfire installation to start the server: "#/openfire start"

Setup Database

- Create the database for Openfire using the openfire_mysql.sql schema file: "cat openfire_mysql.sql | mysql"
- Launch Openfire and use a web browser to connect to the admin console. The default port for the web-based admin console is 9090. If you are on the same machine as Openfire, use the following URL: http://127.0.0.1:9090.
- In the Openfire web-based setup tool, use the following values, where [YOUR_HOST] and [DATABASE_NAME] are the actual values for your server (in many cases localhost is a suitable value for [YOUR_HOST] when your database is running on the same server as your webserver): "driver: com.mysql.jdbc.Driver", "server: "jdbc:mysql://[YOUR_HOST]/[DATABASE_NAME]"

Setup Init Service

- If you would like to install Openfire as a service, one script is provided in the bin/extra directory: openfired.
- Before running this script make sure $OPENFIRE_HOME/bin/openfire is executable by the user you want to run openfire: "chmod +x $OPENFIRE_HOME/bin/openfire"
- Copy the script into /etc/init.d.
- Link the script to your default runlevel directory: "sudo update-rc.d openfired defaults"

IM Gateway Plugin

- Download directly inside the administration console. It will automatically be installed.
- The plugin is configured via the "Gateways" sidebar item located in the Openfire Admin Console. You can enable individual transports via the "Settings" sidebar item, and add new registrations/view existing registrations from the "Registrations" sidebar item.
**Essential Configuration**

- There are several steps to config to ensure the openfire is work with our project.
- 1. Log in openfire as an admin
- Enable external component with port:5275 and secret:cct.
- Either set Allowed to Connect to Anyone or set a Whitelist of the subdomain allowed
- (Note that subdomain and secret should be the same as in the gateway configuration file: xmppservers.xml. See Gateway deployment instruction).

**A2: Installation Instructions for Gateway**

**Extract Package**
- If you want to run it under windows, extract it and move the gateway folder under any directory
- Under Linux system, use "unzip gateway.zip", "mv gateway [anydirectory]"

**Database**
- Create a database, simply write "CREATE DATABASE [databasename]" at the mysql prompt and change the [databasename] to anyname you like to call the database, like "gateway_db"
- Create a table in the database, write the following to the mysql prompt and change the [databasename]: "CREATE TABLE `[databasename]`.delay_message` (`id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT, `message_from` VARCHAR(45) NOT NULL, `message_to` VARCHAR(45) NOT NULL, `via` VARCHAR(500), `content_type` VARCHAR(45) NOT NULL, `body` VARCHAR(1000) NOT NULL, `content_encoding` VARCHAR(45), `content_length` VARCHAR(45), `call_id` VARCHAR(200) NOT NULL, PRIMARY KEY (`id`)) ENGINE = InnoDB;"
- If the client responses with Query OK, you have done with the database setting

**Configuration**
Under installation directory gateway/ there are several configuraion files

1. dbconfig.xml
2. protocols.xml
3. transformers.xml
4. sipconfig.xml
5. xmppservers.xml

- **dbconfig.xml**: configuration about MySQL database
  - `<connection_url>` is the connection string of MySQL database
  - `[host]` is host adress of MySQL deployed
  - `[port]`is port number of MySQL
  - `[databasename]` is the name of database you set
  - An example of `<connection_url>` is like this: `jdbc:mysql://localhost:3306/gatewaydb`
  - `<username>` is the username of your MySQL database, like root
• `<password>` is the password of your MySQL account, like 1234

```xml
<?xml version="1.0" encoding="UTF-8"?>
<connection>
  <connection_url>
    jdbc:mysql://[host]:[port]/[databasename]
  </connection_url>
  <username>username</username>
  <password>password</password>
</connection>
```

• `protocols.xml`: configuration file to set protocol user want to use in the gateway *(If you use XMPP-SIP gateway only, leave it default)*
  
  • `<protocol>` is a single protocol
    • `<protocol-name>` is the name of the protocol, like "SIP"
    • `<protocol-class>` is the java class name of the implemented protocol, like "se.uu.it.cct.gateway.SipLayer"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<protocols>
  <protocol>
    <protocol-name>protocol name 1</protocol-name>
    <protocol-class>class name 1</protocol-class>
  </protocol>
  <protocol>
    <protocol-name>protocol name 2</protocol-name>
    <protocol-class>class name 2</protocol-class>
  </protocol>
</protocols>
```

• `transformers.xml`: configuration files to set transformers *(If you use XMPP-SIP gateway only, only change the gateway domain)*
  
  • `<gateway-domain>` is the adress of the machine where the gateway set to, like "130.238.15.193"
  • `<transformer>` is a single transformer will be used in transform between different protocols
    • `<transformer-name>` is the name of the transformer, like "SIPtoCPIM"
    • `<transformer-class>` is the java class name of the transformer, like "se.uu.it.cct.gateway.util.SIPtoCPIMTransformer"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<translation>
  <gateway-domain>gateway domain</gateway-domain>
  <transformer>
    <transformer-name>name 1</transformer-name>
    <transformer-class>class name 1</transformer-class>
  </transformer>
  <transformer>
    <transformer-name>name 2</transformer-name>
    <transformer-class>class name 2</transformer-class>
  </transformer>
</transformer>
```
<transformer>
  <transformer-name>name 3</transformer-name>
  <transformer-class>class name 3</transformer-class>
</transformer>

<transformer>
  <transformer-name>name N</transformer-name>
  <transformer-class>class name N</transformer-class>
</transformer>

sipconfig.xml: configuration file of sip protocol layer
- `<stack-name>` is the name of sip stack, set as "CctGateway" if you use the siplayer we provide
- `<gateway-ip>` is the address of the machine the gateway deployed on, like "130.238.15.193"
- `<gateway-port>` is the sip port gateway use, like "5060"
- `<outbound-proxy>` is the sip server the gateway connect to, the server can be openser or others, the setting is like "130.238.15.239:5060"
- `<trace-level>` leave it as default
- `<server-log>` leave it as default
- `<debug-log>` leave it as default

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sip>
  <stack-name>CctGateway</stack-name>
  <gateway-ip>130.238.15.193</gateway-ip>
  <gateway-port>port</gateway-port>
  <outbound-proxy>proxy</outbound-proxy>
  <trace-level>32</trace-level>
  <server-log>gateway.log</server-log>
  <debug-log>gateway_debug.log</debug-log>
</sip>
```

xmppservers.xml: XmppServers setting configuration file, since xmpplayer can connect to several Jabber server
- `<server>` is one XMPP server the gateway connect to
  - `<server-name>` is the url address of the XMPP server, like "parthenon.d2dx.com"
  - `<server-port>` is the port of the XMPP server used to establish a component-server connection with our gateway, like "5275"
  - `<server-secret>` is the secret XMPP server set when establish a component-server connection with our gateway, like "cct"
  - `<subdomain>` is the sub domain used, as address of the gateway, like "sip"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<servers>
  <server>
    <server-name>server_url 1</server-name>
    <server-port>port 1</server-port>
    <server-secret>secret 1</server-secret>
    <subdomain>subdomain 1</subdomain>
  </server>
  <server>
    <server-name>server_url N</server-name>
    <server-port>port N</server-port>
    <server-secret>secret N</server-secret>
  </server>
</servers>
```
Run

- Goto the directory gateway installed and run "java -jar Gateway-JainSip.jar"

A3: Installation Instructions for Website

- Download the fuatara website package from http://fuatara.org/downloads/fuatara_web_1.1.zip
- Unzip it to the destination directory.
- Create a database in the target MySQL server.
- Set up the Database tables by executing the db_schema.sql file.

To get the website to work correctly now, two files need to be edited.

General Configuration

This sets up several variables that allow the website to do accurate links and store session information. Edit the following file:
/system/application/config/config.php

and set the following values:

- $config['base_url'] = "/";
  
  Set this to the base URL of the Fuatara website, if it is different from the root. Remember the trailing slash!
- $config['cookie_prefix'] = "";
  
  Set this to a string using only lowercase latin characters (a-z) that represents your domain and is relatively unique.
- $config['cookie_domain'] = "";
  
  To increase security, set this to the domain of the website for increased security

Database Configuration

Now the database settings need to be added to the configuration. Edit
/system/application/config/database.php

and set the following values:

Default

This references the database to be used by the website and notifications servers.

- $db['default']['hostname'] = "localhost";
  
  The MySQL hostname.
- $db['default']['username'] = "";
  
  The MySQL username.
- $db['default']['password'] = "";
  
  The MySQL password
Login
Set the same values as above, but this time to the login information used for the Openfire installation (see above).

**A4: Installation Instructions for Notification Servers**

The following instructions refer to version 0.5 of NotifServers (r996).

- Unzip to any directory.

The NotifServers package comes with all libraries required for successful execution. All that needs to be done is some configuration.

**Configuration**

All servers use the standard CCT Config file, an example of which can be located at [http://trac.fuatara.org/browser/doc/notif/cct_config.txt](http://trac.fuatara.org/browser/doc/notif/cct_config.txt).

For a single-computer setup, all that needs to be done is to set up the database section (this must match the settings entered in the database) and the email sending section. Please note that using local sendmail is unsupported; we require SMTP service.

Once configured, you can optionally place the config file in the default search path (/etc/cct_config.txt).

**Launch**

To launch the servers, simply run

```
./start.sh ./cct_config.txt
```

Or if you put the cct_config.txt in /etc, you can do simply

```
./start.sh
```

**Appendix B: User Guides**

**B1: Mobile Fuatara Client**

**Installation**

- Load the jar file into the mobile phone using bluetooth, an usb cable or infrared.

**Instructions**

**Login**

Open the Fuatara client. It should be under the applications or games folder. Fill in your username and password. Optionally check the auto login option if you want to be automatically logged next time you
open the client.

**Third Party Protocol Settings**

To introduce your login information for those protocols, press the menu button and select AIM/ICQ/MSN.

**MSN Login**

**MSN Login Info**

**Server Settings**

To change your server information, choose Settings from the main menu. Then choose Server. You can choose the domain and port you wish to use, and specify whether or not you wish the application to auto-connect upon startup.

**Server Settings**

Alternatively, you can choose Server when you are at the login screen to be given the same options.

**Chat**

**Send Message**

Scroll down to the contact on your list that you wish to send a message to. Select Chat from the menu. In the chat window, scroll down to the message field. Write your message. Choose OK from the menu, then choose Send from the menu to send the message. It will be displayed on the screen along with the messages sent to you by that contact. Choose Back from menu to return to the main screen.

**Sending Message**
**Forward Message**
Press the forward command of the menu. Choose one or several users in the new screen and press forward to send the message to those users.

- Forward Command
- Forward List

**Open URL**
You can open a url embedded in a message choosing the Go Url option in the submenu.

Open Url in a browser

**Change Status**
From the main menu, choose Status. From the status screen, you can choose which custom message will be shown to your contacts, set your status to Online, Away or Do not Disturb. Choose OK to accept your new settings, or choose Back to leave the status screen while keeping your original settings.

**Log to Other Networks**
From the main menu, choose Status. Choose whether to log in to MSN, ICQ and AIM with your id and password for that respective protocol.
Add a New Buddy

To add a new contact, choose Add from the main menu. Enter which protocol the contact is using (Jabber, MSN, AIM or ICQ). Enter the contact's id, according to the protocol used. It is also possible to name a SIP contact on the fuatara side by using <user>%<fuatara sip server ip>@sip.<OpenFire domain>. For example the contact’s name is joe, OpenSer is running at 130.238.15.169, and OpenFire is at fuatara.com then your contact’s address will look like this: joe%130.238.15.169@sip.fuatara.com.

You may be asked whether the other user may add you to their contact list. Answer Yes or No.

- Add new buddy
- Add new buddy

Delete User

Scroll to the contact you want to delete in the contact list. Choose Delete from the menu. Answer Yes when asked for confirmation that you want to delete this contact.

- Delete buddy
- Delete buddy

Notifications

Read Notification

From your list of contacts, choose the Notifier. You will see a list of notifications. Scroll to the one you want to see and choose Read from the menu. You will be shown a brief description of the post you have been notified of. To return to the notifier screen and choose a new notification to read, choose Back from the menu. From the notifier screen, choose Back to return to the main screen.

- Notification received
List of notifications

**Load Image**

When you press one of the notifications on the list, a new screen will appear showing the content of that notification. It's up to the user to load the image from the website (in case is embedded in the notification). To load it, you must click the option Load Image on the submenu.

**Notification Reader**

- Notification received

**Open URL**

To start the web browser and go to a post you have been notified of, choose Go to URL from the menu when looking at a specific notification.

**Forward Notification**

When viewing a notification, choose Forward To from the menu. On the list that appears, mark the contacts you want to forward it to. Choose Forward from menu to forward the notification and return to viewing the notification. Choose Cancel from menu to return to viewing the notification without having forwarded it.

- Forward Notification
Delete Notification
On the notification screen, scroll to a notification you want to delete. Choose Delete from the menu to delete. Alternatively, choose Delete All to delete all currently stored notification.

Blogging

Create Post

Select Menu and choose Blog in the menu that appears
The post entry form is then shown. Here you can enter
a text message. In the sample illustration the user wants to post a "Hello!" message to her blog.

Post to your blog
It is possible to add media stored on your phone or on a memory card to your blog post. To do this select Menu and choose FileBrowser.

Add Photo

- It is possible to take a photo and post it to your blog.

To do this select Menu and choose Add Photo. You will be taken to a photo preview screen. Choose Large or Small from the menu to set the resolution for taken pictures to high or low. Click the button on the side of the mobile (or alternatively the FIRE button) to take a picture. Choose Send to put the new image file in the blog post. At any time, choose Exit from the menu to leave picture-taking mode without saving a picture.
Add Photo

**Add Audio**

It is possible to add recorded sound to your blog post.

To do this select Menu and choose Add Audio

Add Audio

**Add Video**

To do this select Menu and choose Add Video. You will be taken to a video preview screen. Click the button on the side of the mobile (or alternatively the FIRE button) to start the recording. Click again to stop the recording. Choose Send from the menu to put the new video file in the blog post. Any any time, choose Exit from the menu to leave video-capture mode without saving a video.
**B2: Using The Fuatara Web Site**

1 **Registering on the Fuatara site**

The webpage for registering a new account is located at: www.fuatara.com.

In order to register, select Register in the Main menu on the site. Username and Password is required, Email is not required, but you won't get e-mail notifications if not specified. It can be specified later in My profile, available in the main menu when logged in. Registering will automatically create a jabber account with the same username and password on the server that you can use with a jabber client (more on that later).

2 **Logging In**

Log in by selecting Login in the Main menu. Fill in the required fields.

3 **Subscribing to Blogs and publishing new Blogs**

To subscribe to a blog from the Blogs menu select either search or most popular. Click on subscribe for the blog of your choice. Make sure the kinds of notifications you want are checked (and those you don't want should not be checked). You may add filters, each filter is for that type of notification only, in order to get a filter you must specify wither it is for title, contents or both. If exclusive filter is checked then the word given must NOT appear in order for the notification to be received.

To publish a blog select add blog. That blog must have our plug-in to be able to send notifications. The Blog name and password given here must be the same as defined in the options of the plug-in. The remote posting information is only required in order to quickly post to your blog from our mobile application, and it will only go to whatever blog is selected as default (done from My Published). That information is username and password for a user of the blog that it can post as.

4 **Rss Feeds**

Subscribing to a rss-feed is in many ways similar to subscribing to a blog. Either listing feeds using the Search or Most Popular menu. However rss-feeds are not published, so any user may decide to add any rss-feed (by the use of Add Rss feed) though remember that the URL given there is the URL of the actual feed.
5 Buddies

**List buddies** will list all the buddies you have to your jabber account. **Add buddy** is used to add a buddy obviously, nickname is simply a nickname you wish to refer to that buddy as while id is the buddies id in the form of username@server (unless the buddy uses another protocol then jabber such as SIP, in which case it is username@gateway). Protocol is what protocol the buddy uses, jabber is default.

6 Log Out

To log out simply select **Log Out** in the Main menu.

7 Notifications

Notifications are received as soon as a Blog page or a RSS feed which you have subscribed to is updated.

7.1 Instant Message Instant Message notifications are received in you're logged in IM client using the same account as which you use logging in to the Fuatara site. A few sentences of the Blog entry or Rss feed will be shown as well as the title and an URL (link) to the actual site.

7.2 Widget The Notifications received are presented to the user in the form of a "bubble" appearing on the screen. This appearance is generated by OS-native applications (see below) which present users with an appealing interface that they can configure. When the notification appears, the title will be included in boldface, and an excerpt of the notification text presented to the user, giving him/her an overview of the information presented. The notifications can be left-clicked on. Doing so will lead the user to the webpage that caused the notification to happen. This allows the user to quickly access important data. The notifications can also be right-clicked for them to be instantly dismissed, without any action taken. This allows them to be relatively unobtrusive to a user's workflow.

The Fuatara Client is dependent on **Snarl** for Windows and Growl for Mac OS X (Growl will be installed during the client installation. These are system-wide notification agents that allow users to manage notifications from several sources at the same time. The desktop client (widget) itself can be downloaded for **Mac** and for **Windows**.

7.3 Email Email notifications are sent to the email address which you entered during the registration on the site. The email address can be edited under the Main/**My profile** menu.

8 Mobile Client

The mobile client is called Fuatara and can be downloaded from our website **fuatara.com**. With the mobile client you can send and receive instant messages from you friends, receive notifications if you have subscribed to any blog or rss feed and you can post on your default blog page (choose default blogpage under **My published** blogs).

9 Wordpress Plugins

In order for a blog to send notifications it needs our Fuatara Notifications plugin for wordpress blogs. It is activated as any other plugin, do note that in options it will add options for Fuatara Notify. Weblog Name and Weblog password there MUST be the same as is given for the blog on the Fuatara site for it to work. Fuatara server is simply to which server it should send it to (and should by default point to the right one).
**B2: Adding a Fuatara contact in SipCommunicator**

**Login**

Create New Account

Using SipCommunicator it is possible to talk to other fuatara users which are online on their mobile. First you choose to create a new account. Fill in your user name, password and adress of SIP server(for example, OpenSER server).

![Registration Wizard](image1)

Add Buddy

Create Group

Before adding a new contact to your buddy list is necessary to create a new group. Use the create group wizard to create a new group.

![Create Group](image2)

Add Fuatara Buddy

To add someone not using SIP communicator you will have to fill in your contact address following the below rules:

\(<\text{user-name}>\@\text{jabber server domain}@<\text{gateway ip}>\)

For example, you have a account in Jabber server: a_user@fuatara.com, a gateway with adress 130.238.15.193 and Open Fire running at fuatara.com, the contact address should be like this: a_user@fuatara.com@130.238.15.193.

![Add contact wizard](image3)