A Micro-Crowdsourcing implementation: the Babel Software Project

Chris Exton[1,2], Brendan Spillane[2] , Jim Buckley[1,2]
[1]Centre for Next Generation Localisation,
[2]Department of Computer Science and Information Systems,
University of Limerick

www.cngl.ie
www.localisation.ie
chris.exton@ul.ie; spillane.bren@gmail.com; jim.buckley@ul.ie;

Abstract
Today a lack of access to localisation resources and the cost of these resources contribute significantly to the digital divide. Therefore, finding ways to reduce cost and increase the rate at which software can be localised must be given high priority. While technologies such as Translation Memory and Machine Translation represent a major leap forward in improving this rate, they are, however, viewed mostly as enabling technologies and will not replace the human translator in most cases. This means that, the ability to acquire and manage an extremely large pool of human translators/post-editors is a major stumbling block in improving the rate at which software can be translated, as part of localisation. A Micro-Crowdsourcing architecture, proposed by Exton et al. (2009), could significantly reduce cost and increase capacity within the localisation space. For this reason the authors have created an exploratory implementation based on this architecture to investigate its feasibility.

This paper describes the exploratory implementation of the Micro-Crowdsourcing architecture called Babel Software. This contains two major parts, the development of the Babel Client Library and the Babel Server. The development process was demanding, yet found that the development of full Micro-Crowdsourcing should be feasible. It also found that, although it was possible to retro fit an existing application with the Babel architecture it is not advisable and so Babel should be incorporated into a client application's design during the initial development phase. An extremely important aspect of the implementation was based around the exploration of management tools on the Babel Server. These tools can be used by a project owner managing the localisation community. This helped provide great insight into the infrastructure which the Babel Server should provide in future implementations. Babel Software is an open source implementation which can be found on Souceforge.net. It is hoped that this exploratory implementation will spawn future releases; which will see Micro-Crowdsourcing complement other technologies in the localisation workflow. Together, these technologies can further reduce cost and increase the rate at which software is localised in an effort to reduce the digital divide.

Keywords: *localisation, digital divide, micro crowdsourcing, real time localization, micro-crowdsourcing, babel, open source, java, server*

Introduction

Most commentators agree that the concept of localisation refers to adapting a product to meet the language, cultural and other requirements of a specific target market or "locale" (Collins 2001; Ebben and Marshall 1999). Localisation in software systems involves more than simply translating the content exposed to users of such systems. It also involves adaptation to the local culture (time and date formatting, Number format, cultural conventions, etc…). However, language quality is fundamentally important, as content constitutes the core of these final products (N. Fernández 2000).

Today, there is great demand and value placed on localisation resources. Both commercial organisations and non profit organisations alike require these resources to access different locales. In the commercial sector, large US companies have found that income generated in foreign markets exceeds 65% of their total income (Asnes 2009).

Many of these companies produce or distribute software.

Non-profit or open source projects such as Open Office, also place great emphasis on trying to translate their software into as many languages as possible. There is great value to both types of organisations in being able to localise this software, albeit for very different reasons. Despite this, there is a significant gap between the locales that these organisations might wish to access and what is actually realised. This is highlighted by the fact that most digital information is presented in languages used by countries in the northern hemisphere (Schäler 2004). The lack of access to localisation resources and the cost of these resources is undoubtedly a significant contributor to this reality. This inability to access other locales has a negative effect on society as a whole and contributes significantly to the digital divide. Therefore, finding ways to reduce costs and increase the rate at which software can be localised must be given high priority. Computer automation is an area which provides great promise in this search for reducing costs and increasing capacity. Automating the localisation process is a difficult undertaking due to the inherent complexity encountered during the localisation process. Despite this, technologies such as Translation Memory and Machine Translation represent a major leap forward in improving the rate at which software can be translated as part of localisation. Significantly, however, they are viewed mostly as enabling technologies and will not replace the human translator in most cases. This means that, the ability to acquire and manage an extremely large pool of human translators/post-editors, is a major stumbling block in improving the rate at which software can be localised.

Paper one in this series described an architecture created by Exton et al. (2009) which appears to be well placed to tackle the issue of how to acquire and manage a large pool of translators effectively. The model itself will enable the users of an application to translate the interface while they use it. The user could make a change to UI text, for instance by ctrl-clicking on the text. A popup would then appear allowing the user to update the text. The translations made by the users would then be sent back to a central server where they would be aggregated and analysed. Other users translating the same elements could view the translations from the community members. Each translated element would have an associated rating based on whether or not other community members agreed with it. At this point, there were a number of possibilities as to how the central server can choose to roll-out updates for the software. One possibility proposed by Exton el al. (2009) involved having the Crowd Sourced translations reviewed by a 'trusted' translator. Essentially, this could correspond to post-editing. Once the translations were approved, a new release could be rolled out which would update the UI elements for all the application users.

This Micro-Crowdsourcing architecture could significantly reduce cost and increase the capacity within the translation element of the localisation process. For this reason, the authors created an implementation based on this architecture. This paper describes the implementation, which has the following aims:

- To investigate the feasibility of implementing this type of architecture.
- To investigate whether Micro-Crowdsourcing can be easily applied to existing applications.
- To explore the implementation of tools which could assist in the management of the 'crowds' localisation activities (Stretch goal)
- To identify key areas which future implementations should address.

The implementation undertaken for the purpose of this paper is called Babel Software. The name for the implementation is taken from the ancient city of Babel in the land of Shinar in which the building of a tower (Tower of Babel) intended to reach heaven, was dedicated to the glory of man. God, displeased with the builders' intent, came down and confused the people's languages and scattered the people throughout the earth. Babel represents confusion and an inability to communicate. It is hoped that Babel Software can provide a means from which to minimise this confusion, by providing toolkits and services, which can help bridge the digital divide.

The Babel Server implementation as described here, does take a slightly different view than that put forward by Exton et al. (2009). Although this is software which could be used by commercial organisations to reduce localisation costs, the implementation of the Babel Server is designed specifically with the open source community in mind. The vision is to have a central server which could be used by a large number of open source communities to manage their localisation efforts free of charge.

Babel Software can be broken down into two distinct areas, the Babel Libraries and the Babel Server. The Babel client libraries provide localisation tools which program developers can easily incorporate into their applications. These tools will enable users to localise content from within the application. The Babel Libraries can submit translations to a central server as well as request a list of recommended translations. The Babel Server acts as the central hub to co-ordinate localisation efforts by servicing client requests. The Babel Server also provides essential tools to manage individual projects.

The Babel Libraries will be utilised by projects wishing to localise their software using Micro-Crowdsourcing. Note that throughout this paper, applications which use the Babel Library will be referred to as the Client. From the client's perspective, once they have included the Babel Library as part of their implementation and the software has been internationalised, the localisation process can now begin using Micro-Crowdsourcing. The client owner can then register the project on the Babel Server. Once this is done the client application can be distributed to a community of users by the client owner. Note that the client application will also be available on the Babel Server website. Once at this stage, users operating the application can translate various parts of the application as they use it. The community translators may be working with a base English translation for example or possibly a rough pre-translation using Machine Translation and/or Translation Memory. These translations are then shared with the rest of the community via the Babel Server. Once all elements have been translated the project owner can bundle the translations as part of the applications final release.

The next section focuses on describing the architecture of the Babel Client and the functionality it provides. This is followed by a walkthrough of the functionality provided by the Babel Server and its architecture. Then some of the more significant findings of the Babel Software Project will be discussed. This will be followed by a detailed description of the required future work. Finally the paper will conclude with some final thoughts.

Babel Library

Currently, Babel Software contains a single implementation of a Babel Library for Java Swing Desktop Applications. It is hoped that in the future, additional implementations will be created to cater for other languages. This chapter focuses on describing the architecture and functionality of the Java Babel Client which is called JBabelLibrary.

It is important to note that the Babel Library is not a standalone application; rather it provides functionality which other applications can use. To facilitate the development of this project, an open source application called 'Rachota TimeTracker' was customised to work with the Babel Library. This is a portable java application which is used for time-tracking projects. Again, please note that throughout this paper, applications which use the Babel Library will be referred to as the Client.

The architecture diagram in figure 1 is useful as an introduction to the Babel Library and places it in context. The functionality that the Babel Library provides is utilised by a client, such as Rachota TimeTracker. The Babel Library in-turn performs actions on relevant language files on the operating system and communicates with a larger community of translators via the Babel Server.

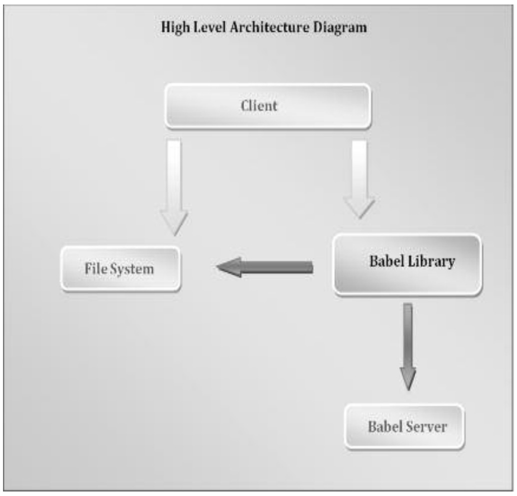


Figure 1: High Level Diagram

Layer 1: Babel Components

A major goal throughout the design of the Babel Library was to make it extremely easy for developers to utilise the library, to the point of adding little or no extra effort to the development process when compared to traditional development. The diagram below is designed to illustrate this point. Developers for instance, normally create JButton's to display clickable buttons on their user interfaces. To implement the Babel Library, developers would instead instantiate JBabelButton's. The JBabelButton class sub-classes JButton and therefore acts in the very same way as a JButton except it provides in-built localisation tools. There are seven Babel components in total, each one sub-classing a different java class. The Babel Library contains many layers which will be described in this section; the Babel Components layer shown in figure 2, is just one such layer. For the most part clients which utilise the library only need to interact with this layer and need not concern themselves with the inner workings of the library.

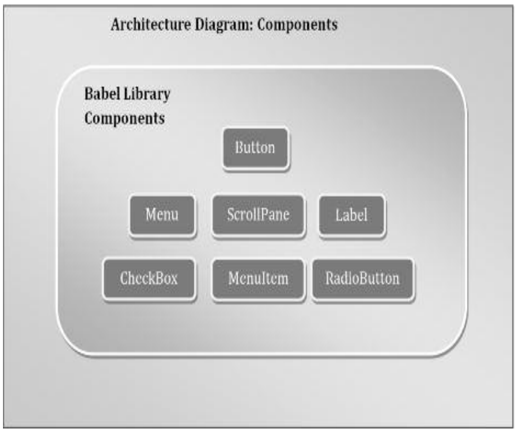


Figure 2: Babel Components (first layer)

Babel Component List
The table below contains a list of the Babel Components and their parent Java Class.

Babel Implementation	Java Class
JBabelButton	JButton
JBabelMenu	JMenu
JBabelScrollPane	JScrollPane
JBabelCheckBox	JCheckBox
JBabelLabel	JLabel
JBabelMenuItem	JMenuItem
JBabelRadioButton	JRadioButton

Table 1: list of the Babel Components and their parent Java Class

Example
Example 1 shows how a client application can implement a Babel button. The JBabelButton is still of type JButton and will behave in much the same manner as the parent class. Each Babel Component has a number of constructors depending on what parts of the component are to be localised. In this example, there is one parameter indicating that it is the primary text within the component which can be localised by the BabelLibrary. In the Babel Software Project, the client application, Rachota TimeTracker, had a language file associated with it on the file system. The parameter passed into the constructor is used to find and translate the correct value within this language file. The parameter is also passed to the Babel Server when requesting and submitting translations. A more detailed explanation of the role of the parameter(s) in the constructors will follow in subsequent sections.

```
JButton exitButton = new JBabelButton("key.exitButton");
```

Example 1: Implementing a Babel button

Each Babel Component has a number of bespoke constructors. Depending on what elements within a component require localisation, a different constructor is called. Currently all components contain two types of constructors.

- Components requiring their main text to be translated.
- Components requiring their main text and tool tip text to be translated.

These constructors contain code which fires events when a component is right clicked.

Layer 2: Babel Interfaces
This sub-section will describe the implementation of the Babel Components and their interaction with the next layer in the hierarchy called Babel Interfaces.

The bespoke constructors contained in the Babel Components provide the link between the first and second layers of the library. Once a Babel Component is right clicked, a call is made to the Interface BabelPopUp. A reference to the calling component itself is passed in the method call.

The BabelPopUp is essentially a popup menu. The popup menu affords the user the opportunity to enter

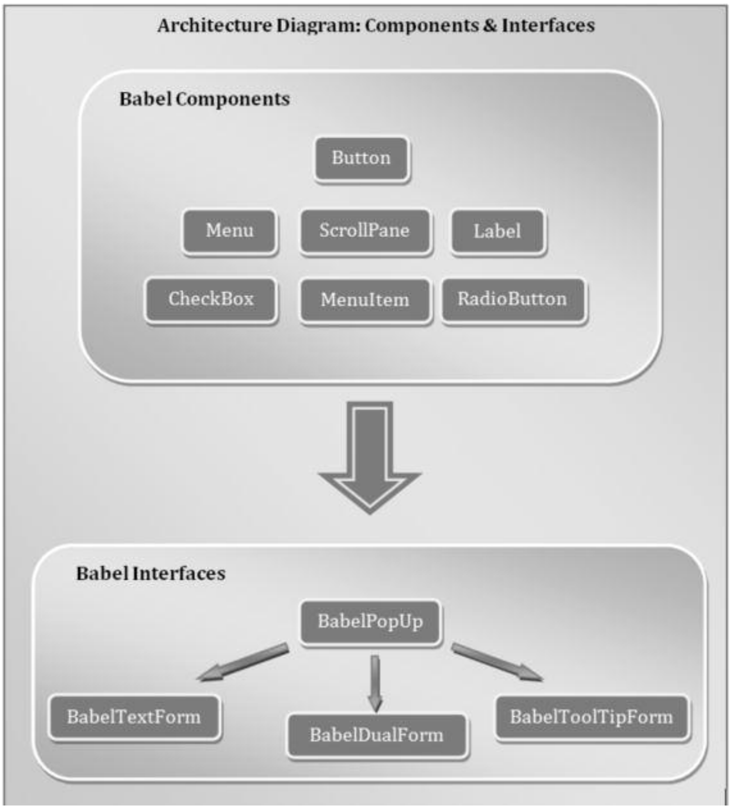


Figure 3: Babel Components & Interfaces (Layers 1 & 2)

a Babel Translation menu. This is illustrated in figure 4: the Rachota Timetracker application has been edited to incorporate the JBabelLibrary. All the buttons labels and menu items seen here, within Rachota, are actually Babel Components. In this instance, the user has right clicked on the button at the top left corner and is presented with a popup. The popup displays an option to 'translate' the component.

Once the user selects 'Translate', the Babel Popup menu disappears and is replaced by one of the Babel Forms. The Popup menu class is responsible for deciding which form is to be displayed; ultimately, this determination is made based on which type of constructor was called when creating the Babel Component.

There are three Babel Forms:

- Babel Text Form, suitable for components which contain only their primary text to be translated.
- Babel Tool Tip Form, suitable for components which contain only tooltip text to be translated.
- Babel Dual Form, suitable for components

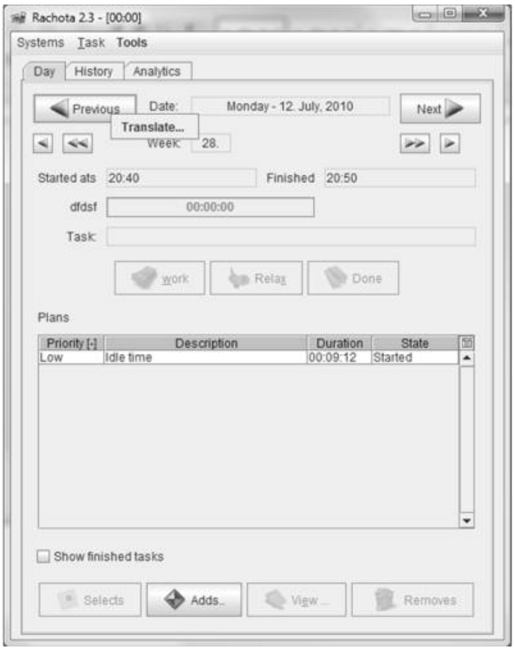


Figure 4: BabelPopUp

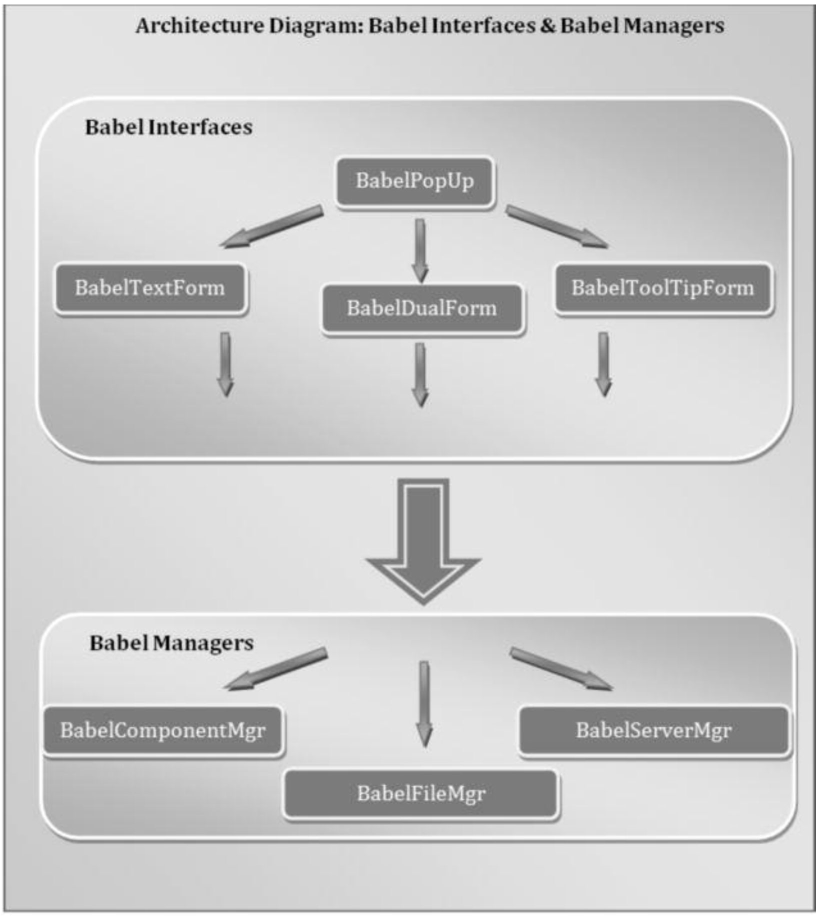


Figure 5: Babel Interfaces & Babel Mgrs (layers 2 & 3)

which contain both their primary text and tooltip to be translated.

Figure 6 shows a screenshot of the Dual Form. As previously stated the form allows translation of both a components primary text and tooltip text. The top half of the form is responsible for primary text translation and the bottom half the tooltip text translation. Both sections have exactly the same functionality.

The sections are simple in their design, allowing the user to trigger two events:

- Firstly, if the user clicks the Retrieve button, the list box to the left displays a list of Babel Community translation suggestions. These translations have an associated rating based on whether or not other translators agree with the suggestion. The Babel Form delegates the responsibility of retrieving the data from the Babel Community to the BabelServerMgr.

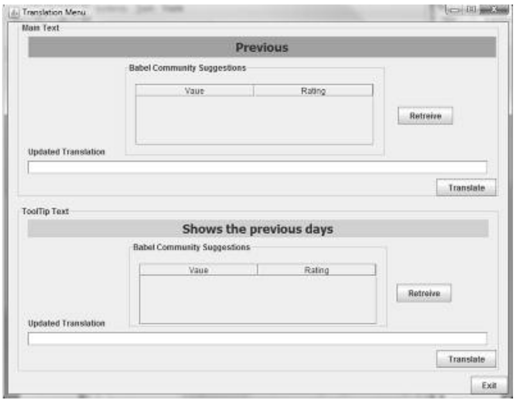


Figure 6: Babel Dual Form

- Secondly, there is an option to allow the user to translate the text. The user can input the updated translation and click the translate button. Once this event is fired, the Babel Form delegates to all three Babel Managers to perform tasks such as updating the User Interface to display the updated translation, writing the updated translation to the language file and finally sending the updated translation to the Babel Community.

The tasks that the Babel Manager performs will be discussed in greater detail in the next section.

Layer 3: Babel Managers & Resources
The three Babel Managers contain the core of the functionality which the Babel Library provides. The managers are responsible for performing tasks on the File System, Application Components & Babel Server.

Babel File Manager (BabelFileMgr)
Clients that run the Babel Library must keep a language file on the system which is used to load text. It is displayed to the user in their selected language. This file and its values are typically loaded when the application starts up. In most cases, the client will store the loaded file in memory as a PropertyResourceBundle for easy access.

When a user submits a translation to the Babel Library, this file must be updated if the changes are to take effect. The Babel File Manager is responsible for this. Once the File Manager has updated the language file, it raises a flag to show that the File has been updated. This prompts the client application to update its PropertyResourceBundle in memory.

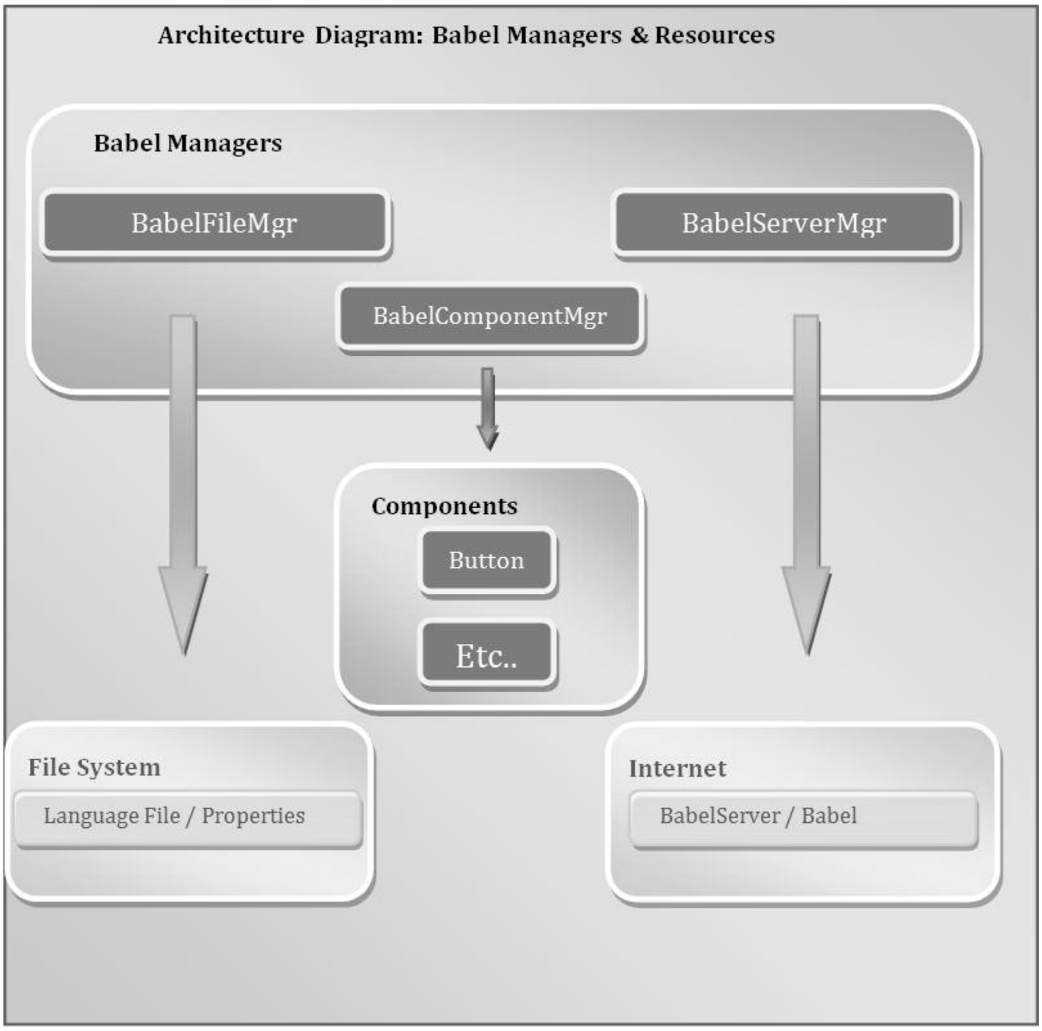


Figure 7: Babel Managers & Resources

Babel Component Manager (BabelComponentMgr)
The File Manager updates text of newly created components and persists the translation(s), however, it does not cater for components which have already been created. When a user has translated a component, this component is viewed as having already been created and its text must be updated dynamically to reflect the user's changes. It is the function of the Component Manager to update existing components.

The only way the Component Manager can possibly update a component is if it has been passed a reference to the object in question. When an event is fired on a component, it passes a reference to the Babel Forms which is forwarded to the Component Manager.

The Component Manager determines what type of object/component it has been passed, i.e. whether it is a JBabelButton or JCheckBox for example. Then the component's text property is updated accordingly.

Babel Server Manager (BabelServerMgr)
Central to the Babel Software is the Babel Server and its ability to co-ordinate and share localisation resources among translators. The Server persists and rates user's translations, while making these translations available to other users. The next section will discuss the Babel Server in greater detail.

The Babel Server Manager is responsible for communicating with the Babel Server. The Babel Server Manager sends data to the server via a HTTP request. Significantly, although the library is in Java, the objects it sends in the request and the objects which are returned from the Server are not Java objects. Instead, both use a language independent data interchange format called JSON. This ensures that the Server will be able to communicate easily with other libraries developed in the future.

The Babel Server Manager performs two key functions for the library:

- Firstly, it can request recommended translations from the Babel Server for a particular component. This function forwards the application name, application language and the component identifier to the Babel Server. The Server then returns a list of recommended translations with an associated rating.
- Secondly, it sends user translations to the server. It forwards the application name, application language, the component identifier and the user's translation to the Server. The Server then responds with a message stating whether or not the translation was processed.

Babel Server
This section describes the design, implementation and functionality offered by the Babel Server. The server is central to the concept proposed by Exton et al. (2009). It is the hub which co-ordinates the localisation activities of a community of translators.

Figure 8 details a high level architecture to place the Babel Server in context, relative to external applications with which it interacts. On the left side of the diagram, a client using the Babel Library is seen interacting with the server. This could represent the Rachota application as seen previously, or any other application implementing the Babel Library. Another significant facet to the Babel Server, which can be seen on the right side of the diagram, is the Babel Website. This website can be viewed like any other, via a web-browser. Its primary aim is to provide online tools from which to monitor and manage the localisation of a project/application.

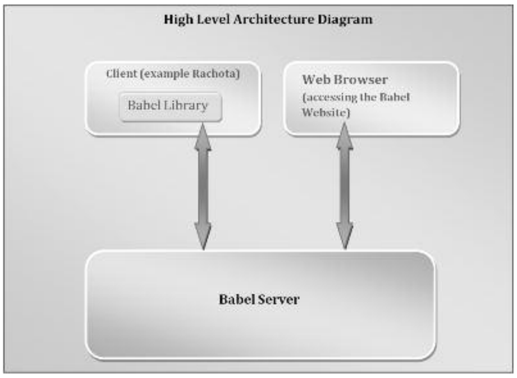

High Level Architecture Diagram
Client (example Rachota)
Babel Library
Web Browser
(accessing the Babel Website)
Babel Server

Figure 8: High Level Architecture Diagram

As can be seen from Figure 8 the Babel Server can be divided into two distinct areas, functionality that services Clients or requests from Babel Libraries and functionality which supports the website. Both sets of functionality will be described in detail later in this section. Firstly, however, a technology overview for the entire server will be presented as both areas of the server leverage much of the same technologies.

Technology Overview
The Babel Server is essentially a web server which utilises the Spring Framework. Irrespective of whether or not a request comes from a client using the Babel Library or whether it is a request to view the website, the architecture outlined in figure 8 processes the request. The Spring Framework was chosen for the Babel Server as it provides enormous flexibility. This is extremely important when keeping in mind that it is hoped that there will be future releases of the server which will build upon the existing implementation. Without the implementation of the Framework, the process of reusing or even comprehending existing code would be made extremely difficult.

Spring Framework
The Spring Framework consists of 7 modules which cater for various aspects of JEE application development. Currently, the Babel Server implements only two Spring modules, however it is hoped that this number will increase in the future. The two modules which the server uses are; Springs Core Container and its MVC Framework. The core container is the primary module within the framework. The core container provides a means to separate configuration settings and dependencies from the code itself. It does this using the Inversion of Control (IOC) pattern. The Inversion of Control pattern or dependency injection, allows you to describe how objects are to be created without creating them and their dependencies directly in application code. This involves specifying the dependencies between objects or components in a configuration file. The container then interprets this specification and is responsible for the creation of objects and their dependencies. (Balani 2005)

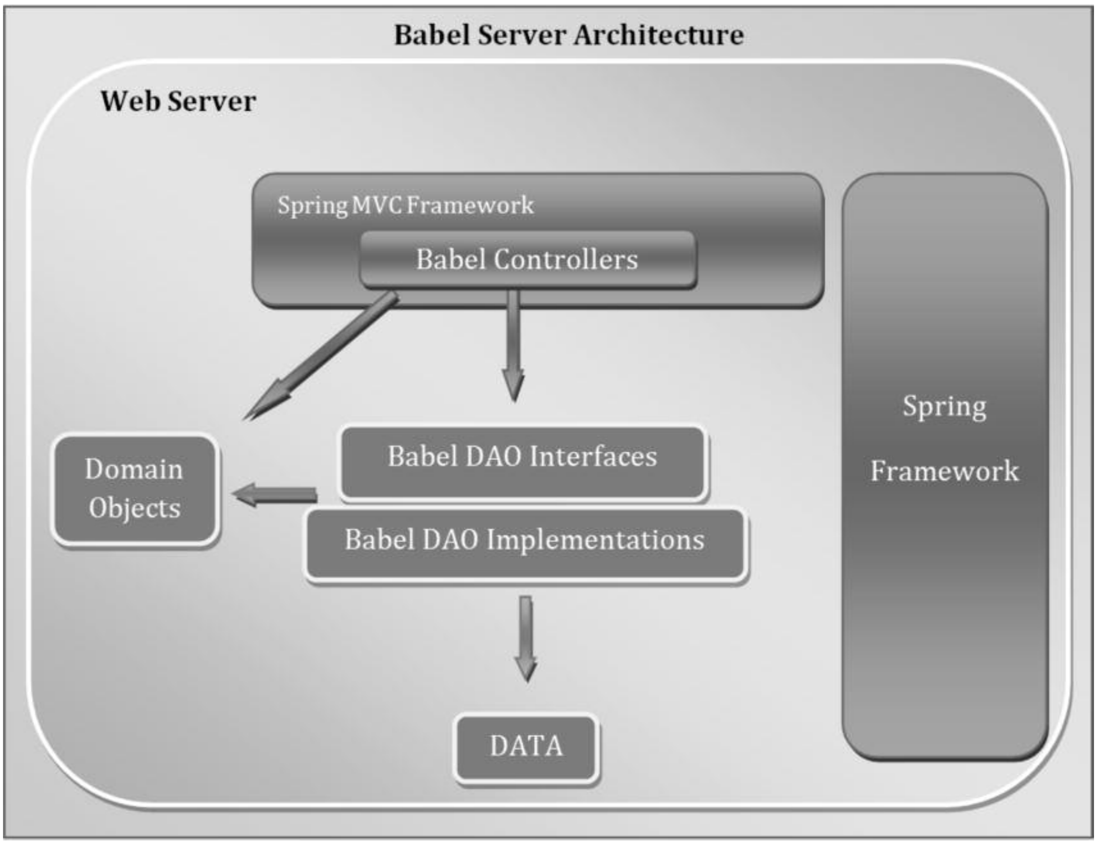


Figure 9: Babel Server Architecture

The Inversion of Control pattern provides a means of separating application logic into layers which can be interchanged and reused if necessary. As mentioned earlier, this flexibility was the primary reason for choosing the Spring Framework. An example of this flexibility in practice can be seen with the Data Access Object layer (DAO) as seen in figure 8. Although the surrounding layers are much more mature, the development of the DAO layer is in its preliminary stages, the existing implementation will be replaced in future releases. The IOC pattern enables the developer(s) to insert a new DAO, without affecting other layers of the application.

The Spring MVC Framework is a module for building web applications. Spring MVC (model-view-controller) uses the MVC architectural pattern which separates business, navigation and presentation logic (Springdeveloper.com n.d.):

- *Controller:* Contains navigation logic and interacts with other application layers like the Data Access Object (DAO) to retrieve business logic. This is illustrated in Figure 8, the Babel Controllers interact with the Babel DAO and domain objects to retrieve business logic.
- *Model:* Contains the data which is required by the View. This is retrieved from the Babel DAO.
- *View:* Is responsible for pulling data from the model and rendering the output. There are two types of views used by the Babel Server, JSON and JSP. JSON is what is returned to clients using the Babel Library and a JSP is returned when accessing the Babel website.

The key benefit to Spring MVC is similar to that of its parent. The separation of logic makes the code much easier to change in the future.

Data Structure

As can be seen in Figure 8 the controllers and DAO utilise the Domain objects. The Babel implementation contains three Domain objects, Project, Dictionary and DictionaryElement. The data structure is designed to capture the various types of data required to describe the communities' translations for all projects:

- An object of type Project could represent an application such as Rachota or Open Office. The class contains simple data types to store the project name, date created and authorisation data. Also, each project can have a list of Dictionaries associated with it.
- Each language that an application caters for would be encapsulated within a dictionary object. The dictionary class contains information on the language/locale the dictionary represents as well as a list of dictionary elements. The dictionary class also provides functionality to search for dictionary elements and update specific element attributes.
- Dictionary elements represent components, such as buttons and text fields in the client application. Each dictionary element contains a number of fields, which store the components unique identifier used by the client application, as well as an associated translation. Significantly, the translation is also given a rating based on the community's approval/disapproval.

Rating system

Contained within the domain objects is the logic for the voting system. The current implementation enforces a very simple system, whereby if another community member agrees with a translation, its associated rating is simply incremented.

Library requests

This sub-section describes the implementation of components which service Babel Library requests. The diagram in figure 9, identifies the main areas of interest and the flow of control with regard to servicing library requests. For simplicity, the diagram below shows only one client library. In reality the server can process requests from multiple client applications.

One of the most important aspects of the diagram below is the fact that the server returns JSON objects to the client. JSON is a lightweight data interchange language; the rationale for choosing this format is closely linked to the future needs of the Babel project. If the development of no other client libraries was envisaged then the server could simply return a Java object which could be easily understood by the client. However other client libraries are expected to be developed which cater for other languages. Therefore returning language specific objects is not appropriate. Using JSON means that the server does not need to change in order to accommodate other client libraries and in turn all client libraries can easily parse responses from the server.

Client applications implementing the Babel Library can request two services from the Babel Server:

- Request a list of recommended translations
- Submit a translation.

The client can request these services by making a

HTTP request and passing parameters including application name and the name of the component to be translated. The web server then directs the Babel Library request to the relevant Spring MVC controller, either the 'requestTranslationController' or the 'submitTranslationController'. The controller then retrieves the 'Model' or the data by delegating to the Data Access Object (see Figure 8). Once the Model has been retrieved the Controller directs the flow of control to the JSON view (see Figure 9).

The Spring Framework requires that all information sent back to the client is sent via a class implementing the Spring Interface 'View'. Although spring bundles implementations for JSP's and XML for example, it does not cater for JSON. Therefore the JSON View and JSON Util classes shown in figure 8 were created during the development process. The JSON View implements the Spring Interface View. It has an associated helper class called 'JSONUtil' which converts the data to a JSON

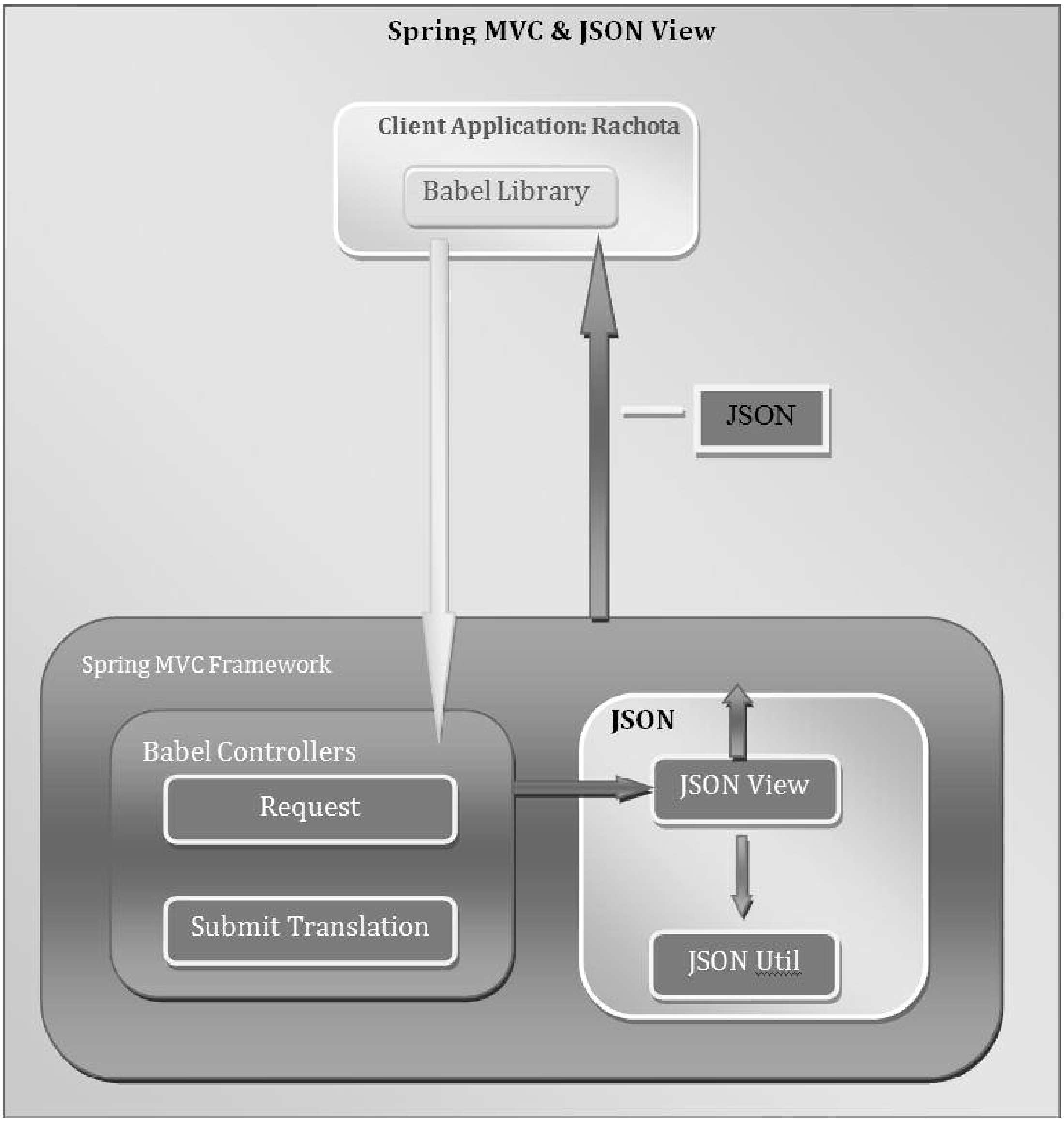


Figure 10: Spring MVC & JSON View

object. The JSON view then forwards the JSON object to the client.

Babel Website

The Babel website has two major driving factors: Firstly the website provides a valuable means through which to advertise Babel Software and subsequently encourage users to join the community. Secondly, the website provides tools to manage projects running on the server. This sub-section begins with a brief description of its implementation, followed by a description of the functionality provided by the Babel Website.

Architecture

The processing of requests from web browsers is much the same as requests from client libraries. There are five Babel Controllers including 'Add Dictionary Controller' and 'Delete Dictionary controller'. The controllers retrieve the 'Model' or the data by delegating to the Babel DAO *(see Figure 8)*. The Babel DAO performs operations such as adding or deleting dictionaries associated with a given project. Once the Model has been retrieved, the Controller instantiates a View object. This process differs from the previous section as a JSP View is used instead of a JSON view. As stated previously Spring does bundle a JSP View with the Framework making this process less time consuming. During the instantiation of the JSP View, it is attached to a specific JSP page such as 'addDictionary.jsp' or deleteDictionary.jsp *(see Figure 10)*. These JSP's are then displayed in the client's web browser as a web page.

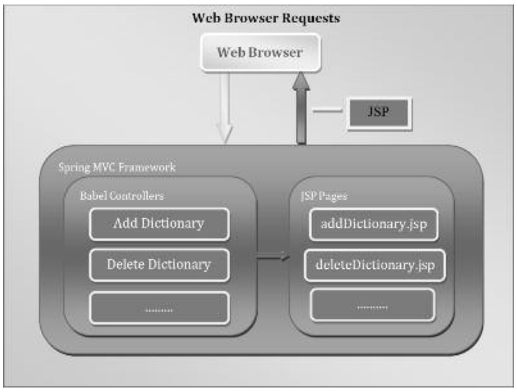
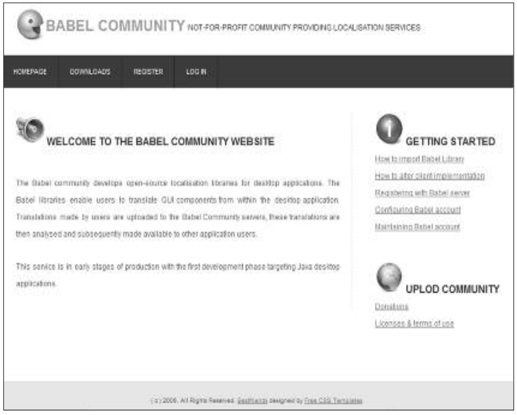


Figure 11: Web Browser Requests

Advertising the Babel Community

The first page or homepage of the website is largely targeted at engineers. It provides a brief overview of what Babel Software is and provides links to resources such as how to configure the Babel Library and manage a Babel account.


BABEL COMMUNITY NOT-FOR-PROFIT COMMUNITY PROVIDING LOCALISATION SERVICES
HOMEPAGE
DOWNLOADS
REGISTER
LOG IN
GETTING STARTED
How to import Babel Library
How to alter client implementation
Registering with Babel server
Configuring Babel account
Maintaining Babel account
UPLOD COMMUNITY
Donations
Licenses & terms of use

WELCOME TO THE BABEL COMMUNITY WEBSITE

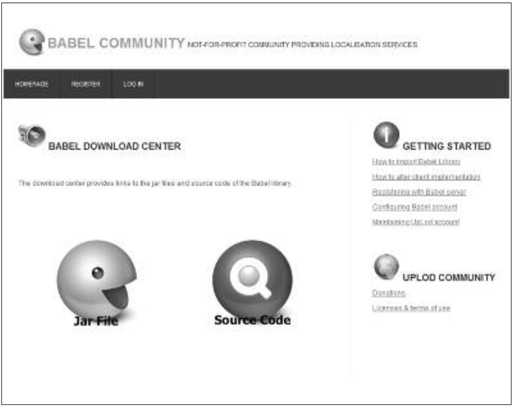
The Babel community develops open-source localisation libraries for desktop applications. The Babel libraries enable users to translate GUI components from within the desktop application. Translations made by users are uploaded to the Babel Community servers, these translations are then analysed and subsequently made available to other application users.

This service is in early stages of production with the first development phase targeting Java desktop applications.

(c) 2008. All Rights Reserved. [illegible] designed by Free CSS Templates

Figure 12: Website Homepage

The Babel Download centre is easily accessible from the homepage. It provides an option to download the Babel Client library (currently only for Java Swing). Also there is a link to the source code which directs the user the Babel Software SourceForge page. Both the Babel Client and Babel Server source code is open-source and is accessible via a subversion repository.


BABEL COMMUNITY NOT-FOR-PROFIT COMMUNITY PROVIDING LOCALISATION SERVICES
HOMEPAGE
REGISTER
LOG IN
Jar File
Source Code
GETTING STARTED
How to import Babel Library
How to alter client implementation
Registering with Babel server
Configuring Babel account
Maintaining Upl od account
UPLOD COMMUNITY
Donations
Licenses & terms of use

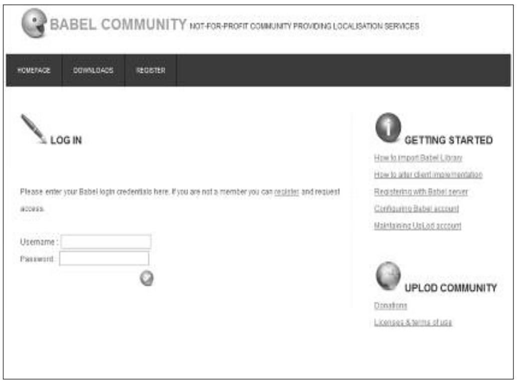
BABEL DOWNLOAD CENTER

The download center provides links to the jar files and source code of the Babel library.

Figure 13: Website Download Page

Project Management/Administration

The Babel Server implementation for the website allows a project administrator for a project such as Rachota to login and manage and oversee various aspects of the community. Figure 14 shows how users can log into the site and enter the Administration area for their project.


BABEL COMMUNITY NOT-FOR-PROFIT COMMUNITY PROVIDING LOCALISATION SERVICES
HOMEPAGE
DOWNLOADS
REGISTER
LOG IN
GETTING STARTED
How to import Babel Library
How to alter client implementation
Registering with Babel server
Configuring Babel account
Maintaining UpLod account
UPLOD COMMUNITY
Donations
Licenses & terms of use

Please enter your Babel login credentials here. If you are not a member you can register and request access.

Username :
Password :

Figure 14: Website Login page

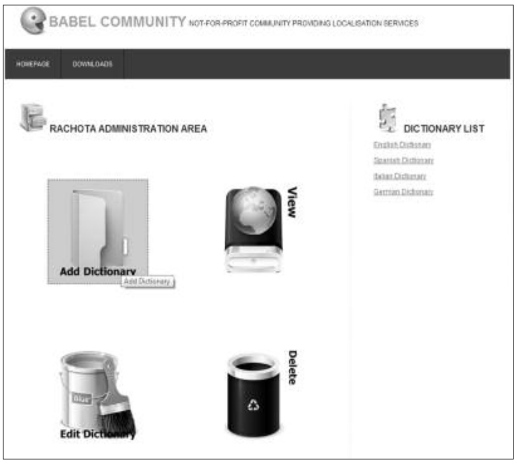
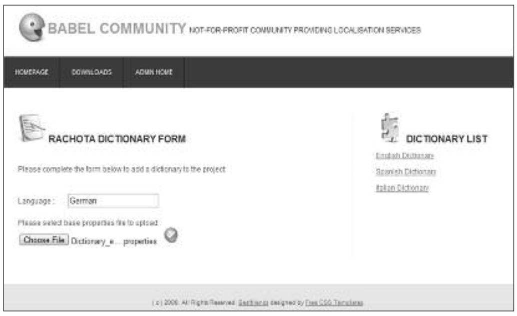


Figure 15: Website Administration area

Figure 15 is a screenshot of the Administration area for an administrator of the Rachota application which was mentioned earlier. In this sub-section we will describe three of the options available to the administrator:

- Add a dictionary
- View Translations/dictionary
- Delete a dictionary.

Figure 16 shows a screenshot of the add dictionary page. In this instance the project administrator is attempting to create a German dictionary for the Rachota application. From here the project administrator can upload a base language properties file. The uploaded properties file could, for example, contain rough machine translations for the language in question. Once uploaded the Add DictionaryController' delegates to the DAO which creates a new dictionary for the project as well as dictionary elements based on the data in the properties file. Note that each element uploaded in the properties file has its rating set to 0. Once this process is complete, clients with the German version of Rachota can submit and request translations to the Babel Server.



Please complete the form below to add a dictionary to the project

Language : German

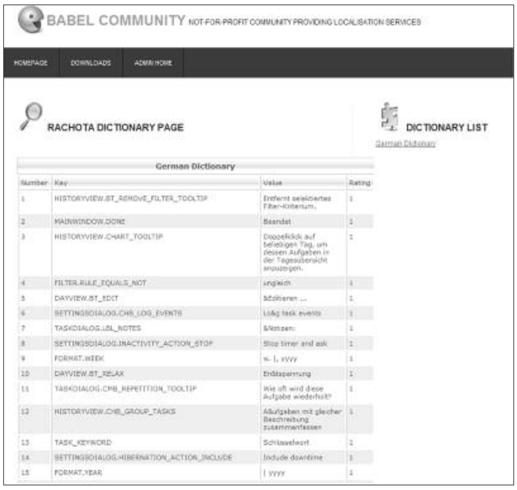
Please select base properties file to upload

Dictionary_e... properties

(c) 2006. All Rights Reserved. [illegible] designed by Free CSS Templates

Figure 16: Add Dictionary Page

Throughout this process, the project administrator can view the communities' progress and can see ratings for the various dictionary elements as seen in figure 17. When the View Dictionary page is requested by the user the 'ViewDictionaryController' delegates to the DAO which retrieves data in the form of domain objects to be displayed to the user. The dictionary information displayed to the user could be an extremely valuable tool in overseeing the translation process.

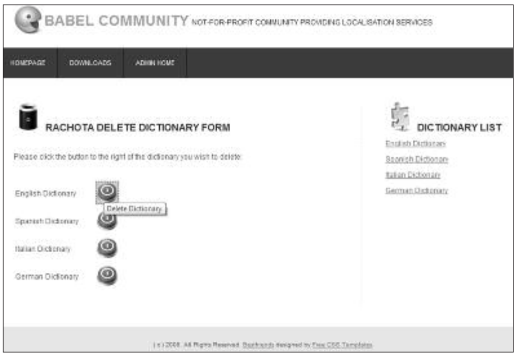


German Dictionary

Number	Key	Value	Rating
1	HISTORYVIEW.BT_REMOVE_FILTER_TOOLTIP	Entfernt selektiertes Filter-Kriterium.	1
2	MAINWINDOW.DONE	Beendet	1
3	HISTORYVIEW.CHART_TOOLTIP	Doppelklick auf beliebigen Tag, um dessen Aufgaben in der Tagesubersicht anzuzeigen.	1
4	FILTER.RULE_EQUALS_NOT	ungleich	1
5	DAYVIEW.BT_EDIT	&Editieren ...	1
6	SETTINGSDIALOG.CHB_LOG_EVENTS	Lo&g task events	1
7	TASKDIALOG.LBL_NOTES	&Notizen:	1
8	SETTINGSDIALOG.INACTIVITY_ACTION_STOP	Stop timer and ask	1
9	FORMAT.WEEK	w. |, yyyy	1
10	DAYVIEW.BT_RELAX	En&tspannung	1
11	TASKDIALOG.CHB_REPETITION_TOOLTIP	Wie oft wird diese Aufgabe wiederholt?	1
12	HISTORYVIEW.CHB_GROUP_TASKS	A&ufgaben mit gleicher Beschreibung zusammenfassen	1
13	TASK_KEYWORD	Schlüsselwort	1
14	SETTINGSDIALOG.HIBERNATION_ACTION_INCLUDE	Include downtime	1
15	FORMAT.YEAR	| yyyy	1

Figure 17: View Dictionary

The final page in the website walkthrough is the delete dictionary page. This allows the project administrator to remove a dictionary from the server by simply clicking on the red button as shown in figure 18.



RACHOTA DELETE DICTIONARY FORM

Please click the button to the right of the dictionary you wish to delete

English Dictionary
Spanish Dictionary
Italian Dictionary
German Dictionary

(c) 2008. All Rights Reserved. Dashboards designed by Free CSS Templates

Figure 18: Delete dictionary page

Discussion

The implementation of the Babel Client Library and Babel Server as described in the previous two sections was deemed to be a success. No significant road blocks were encountered and a great deal of functionality was implemented for the Client Library and Server.

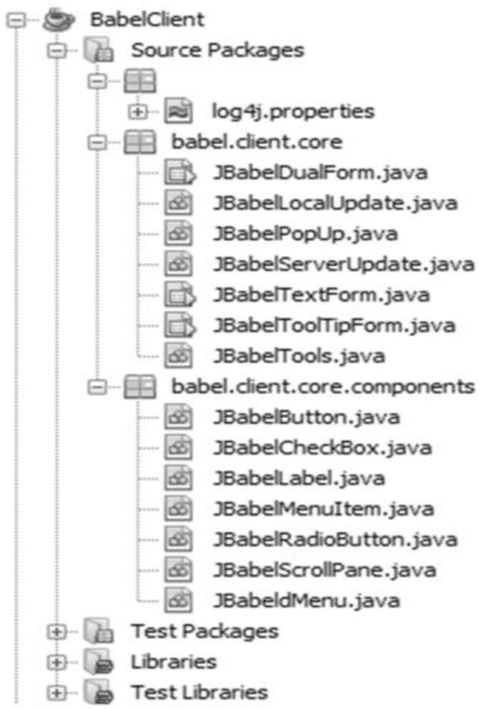
This section will discuss some of the more significant findings and experiences during the implementation process which relate to three of the four goals of this exploratory implementation:

- To investigate the feasibility of implementing this type of architecture.
- To investigate whether Micro-Crowdsourcing can be easily applied to existing applications.
- To explore the implementation of tools which could assist in the management of the 'crowd's' localisation activities (Stretch goal)

Client Library

The implementation of the Babel Client Library for Java was an extremely time consuming process. The implementation of the babel.client.core package (see figure 19) was the most demanding and was of critical importance to the success of the project. In contrast, the implementation of the components package was much less demanding. Therefore, based on the development of the seven Babel components, the expansion of the library itself to include other components should be possible in a shorter timeframe than first expected.

One area where issues were encountered was when attempting to retro-fit an existing application with the Library, as opposed to building an application with the Babel Library in mind. Significant changes were made to the client application Rachota, to facilitate the Babel Library.



```
BabelClient
  Source Packages
    <default package>
      log4j.properties
    babel.client.core
      JBabelDualForm.java
      JBabelLocalUpdate.java
      JBabelPopUp.java
      JBabelServerUpdate.java
      JBabelTextForm.java
      JBabelToolTipForm.java
      JBabelTools.java
    babel.client.core.components
      JBabelButton.java
      JBabelCheckBox.java
      JBabelLabel.java
      JBabelMenuItem.java
      JBabelRadioButton.java
      JBabelScrollPane.java
      JBabeldMenu.java
  Test Packages
  Libraries
  Test Libraries
```

Figure 19: Babel Client Packages & Classes

As is standard with Java Swing applications, Rachota is contained in a jar file which is an executable. Clicking on this file starts the application. On start-up Rachota reads from a properties file which contains translations for a given language. The language file that the application calls is dependent on the locale settings of the operating system on which the application is run. Once Rachota has read in the file it loads the data into a PropertyResourceBundle. Using the PropertyResourceBundle makes it easier for the application to query the data which it read from the properties file. When rendering each screen to the user, Rachota calls on a class called Translator to populate values for parts of the application that have been internationalised.

```
Translator.getTranslation("DAYVIEW.BT_NEXT")
```

Example 2: Rachota calling the Translator class

The translator class queries the Property ResourceBundle in memory, for the correct translation. This implementation presented a number of issues to the development process. Firstly, the properties files or language files are kept within the executable/jar file. This means that the 'Babel File Manager' cannot update the file with translations from the user or other members of the community. Therefore, the language files must be kept outside the jar file so that they can be written to as needed. Rachota code was updated to reflect this. Secondly, when retrieving a translation for an element, Rachota retrieves the translation from the ResourceBundle and not the file itself. Therefore, if any change has been made to the file since the ResourceBundle was created, then the ResourceBundle will not contain up to date information. In order to compensate for this, the Babel Library flags when any changes have been made to the properties file. Subsequently, a Rachota method called Translator.getTranslation(..) was altered; before it reads from the ResourceBundle, it checks to make sure that it is up to date. If the ResourceBundle is not up to date, then Rachota clears all data in the ResourceBundle and re-populates the object with data from the updated properties file. There were a number of other minor issues which had to be overcome with Rachota when trying to utilise the Babel Libraries functionality; however, once these issues had been overcome, it was found that Babel components can be incorporated quite easily into the implementation of a client application. This involves the replacement of code seen below for a JButton with the instantiation of a JBabelButton:

```
JButton exitButton = new JButton( Translator.getTranslation("key.exitButton") );
```

Example 3: Original Rachota JButton code

```
JButton exitButton = new JBabelButton("key.exitButton");
exitButton.setText(Translator.getTranslation("key.exitButton"));
```

Example 4: Modified Rachota JBabelButton code

The text in the constructor for the Babel Component was used to locate the relevant translation in the properties file. Calling the setText simply populates the text of the component with the appropriate translation.

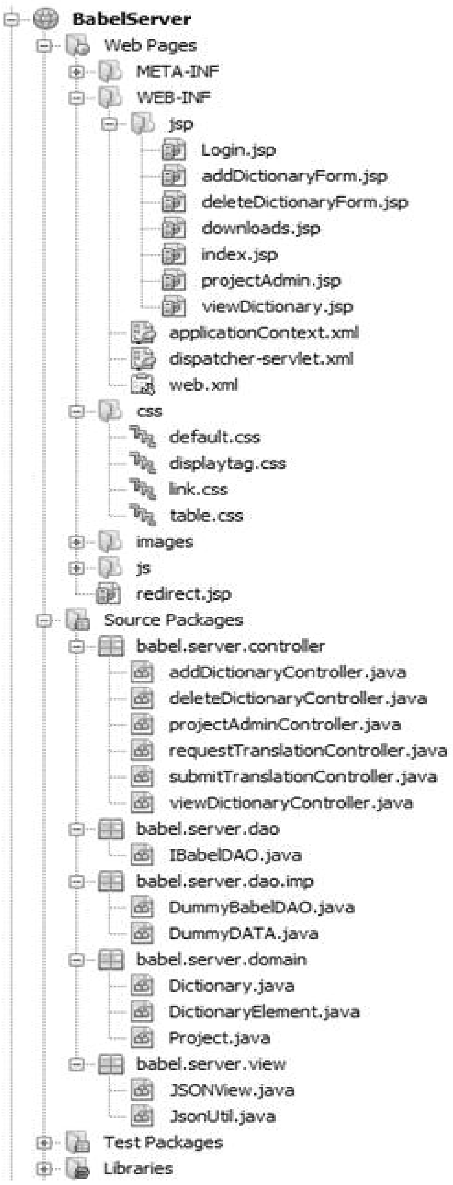
The difficulties faced incorporating the library into the Client application Rachota were not what one might describe as difficult engineering activities, however, they were very time consuming. When keeping in mind that the code base for Rachota is quite small and its internationalisation classes are relatively concise, one can imagine that attempting the same activity with a larger application could be a major engineering activity indeed. This would suggest that the incorporation of the Babel Library into a client application should be done during its development process. At the very least, the design of its internationalisation classes should be conceived with a view to accommodating the Babel Library. This step alone need not necessarily add to the initial development cost and would ensure that the client would be compatible with the Babel Library.

Server

Throughout the development process, the Babel Client implementation was given highest priority. Therefore, the Server was given a shorter timeframe for development than the client.

The primary goal for the development of the Babel Server was to create a skeleton implementation which, firstly, could service client requests from Babel Libraries written in different languages and also serve as the base implementation from which future releases would build upon. Therefore, great thought was given to the overall design of the server and its constituent parts. As was mentioned earlier, the Spring Framework was used in the implementation of the Babel Server; although for smaller projects or once off implementations, the use of such a framework could be seen as needless and labour intensive. Due to the advantages provided by techniques, such as dependency injection, it was seen as essential in ensuring that the current implementation could be built upon in the future. The use of an intermediate data interchange format, via HTTP protocol, is the key to ensuring that the server is compatible with other client library implementations. The concrete classes were then 'wired' together using Spring. Throughout this process, various revisions of classes were introduced and subsequently re-wired into the framework with ease, further highlighting the importance of the framework even at this early stage.

Figure 20 contains a screenshot of the package and class structure of the server. The large number of classes and files created for the Babel Server reflects the high level of productivity during this phase of development. As a result, the development of the Babel Server went far beyond the initial goal of building merely a base implementation. Instead, the server provided a testing ground for ideas and aided



```
BabelServer
  Web Pages
    META-INF
    WEB-INF
      jsp
        Login.jsp
        addDictionaryForm.jsp
        deleteDictionaryForm.jsp
        downloads.jsp
        index.jsp
        projectAdmin.jsp
        viewDictionary.jsp
      applicationContext.xml
      dispatcher-servlet.xml
      web.xml
    css
      default.css
      displaytag.css
      link.css
      table.css
    images
    js
    redirect.jsp
  Source Packages
    babel.server.controller
      addDictionaryController.java
      deleteDictionaryController.java
      projectAdminController.java
      requestTranslationController.java
      submitTranslationController.java
      viewDictionaryController.java
    babel.server.dao
      IBabelDAO.java
    babel.server.dao.imp
      DummyBabelDAO.java
      DummyDATA.java
    babel.server.domain
      Dictionary.java
      DictionaryElement.java
      Project.java
    babel.server.view
      JSONView.java
      JsonUtil.java
  Test Packages
  Libraries
```

Figure 20: Babel Server Packages & Classes

in the discovery of new possibilities. The idea behind Babel Software is that a small community of translators work together to localise a piece of software. The product owner has now been unburdened from a great deal of the 'heavy lifting' associated with this task. Also, the rating system provides an autonomous way of collating and grading the contributions of translators. All of this is key in reducing the cost of localisation. However, this does not mean that a product owner can simply ask a community to translate a product and come back after a few weeks and hope to find a complete translation. The product owner must administer or manage the translation process. The speedy development process, facilitated the exploration of the functionality which the server could and should provide in the future. The functionality which the server should provide, was derived from the vision of creating a Babel Server which would service a large number of open-source projects free of charge, similar to SourceForge.net. As stated in the introduction to this paper, although supporting open-source communities is the primary aim of this development effort, it could also be leveraged by commercial organisations. With this in mind, the development of a website to promote the service, and the development of an administration area containing a toolkit were the focus for this exploratory work.

The vast majority of this exploratory work surrounded the development of the administration area. The ability for a project administrator to log in to the system and perform various management tasks undoubtedly is a very powerful capability. Even during initial testing it was obvious that the ability to add a new dictionary on the fly was enormous. Within seconds the infrastructure could be put in place to support the work of a group of translators. Once in place, the ability to monitor the communities can help gauge the project's progress as a whole and view the quality of translations. This monitoring facility could be an ideal tool for language experts; and could serve as the basis for future implementations which support auditing or approvals of the community's translations.

The aforementioned experiences during the development process for this exploratory implementation have led the authors to following assertions: Firstly a full implementation of the Micro Crowdsourcing architecture appears to be a feasible undertaking. Significantly, the design of the exploratory implementation lends itself to being the primary building block for this full implementation. Also, the author's experiences suggest that attempting to retro-fit existing applications with this Micro-Crowdsourcing architecture could be a time consuming and expensive undertaking. Finally, the exploration of possible management tools for use on the Server by an administrator suggests that they could provide an extremely powerful solution without incurring significant development costs.

Conclusion

Babel Software is an exploratory implementation of a Micro-Crowdsourcing architecture proposed by Exton et al. (2009). The Babel Software Project is an Open source implementation which can be found on SourceForge.net. Babel Software consists of two main parts, the development of a client library called Babel Library and a server called the Babel Server. As part of the development of the Babel Library, it was incorporated into an existing Java Desktop application or client. The development of the Babel Client Library and Babel Server prototypes proved successful. The development process was demanding yet found that the development of the Micro-Crowdsourcing architecture was feasible. It also found that, although it was possible to retro fit an existing application with the Babel Client Library, it was time consuming and not advisable. Therefore, any client application wishing to leverage the Micro-Crowdsourcing architecture should incorporate the Babel Library into its design during the initial development phase. At the very least, the design of a clients internationalisation classes should be conceived with a view to accommodating the Babel Library. A stretch goal of the implementation was to develop management tools for users on the Babel Server. This proved to be an extremely important aspect of the implementation and helped provide great insight into the infrastructure which the Babel Server should provide in future releases.

The implementation of this Micro-Crowdsourcing architecture, has the capability to increase the volume of localised material without necessarily compromising on quality. Although it is not seen as a complete solution to meet the increasing demand for localisation services, it is well placed to complement other technologies in the localisation workflow. It is hoped that technologies such as Machine Translation and Translation Memories together with Micro Crowdsourcing, can combine to dramatically decrease costs and increase the rate at which software can be localised.

References

Asnes, A., 2009. Internationalization ROI. MultiLingual, 20(6), 32-35.

Balani, N., 2005. The Spring series, Part 1: Introduction to the Spring framework. Available at: http://www.ibm.com /developerworks/web/library/wa-spring1/.

Collins R.. 2001. Software Localization: Issues and Methods, The 9th European Conference on Information Systems. Bled, Slovenia.

Ebben, S. and Marshall G.. (1999) The Localisation process: Globalizing your Code and Localizing your Site. http://msdn.microsoft.com/workshop/management/intl/locp rocess.asp (accessed on 6th June 2000)

Elliott, J., 2005. What is Hibernate. Available at: http://onjava.com/pub/a/onjava/2005/09/21/what-is-hibernate.html.

Exton, C. Wasala A., Buckley J and Schäler R., 2009. Micro Crowdsourcing : A new Model for Software Localisation. The International Journal of Localisation, 8(1), 81-89.

Fernández N, 2000 Web Site Localisation and Internationalisation: a Case Study. MSc Thesis, City University London.

MySQL, Why MySQL. Available at: www.mysql.com/why-mysql/.

Schäler, R., 2004. The Cultural Dimension in Software Localisation. Localisation Reader 2003-2004, 5-8.

Smith, B., 2007. A Quick Guide to GPLv3. Available at: http://www.gnu.org/licenses/quick-guide-gplv3.html.

Sourceforge.net, 1999. About Sourceforge. Available at: http://sourceforge.net/about.

Sourceforge.net, 1999. About Subersion. Available at: http://sourceforge.net/scm/?type=svn&group_id=332838.

Sourceforge.net, 1999. Sourceforge Homepage. Available at: http://sourceforge.net/.

Springdeveloper.com, Spring MVC. Available at: http://www.springdeveloper.com/oscon/mvc.pdf.