Greetinq - VoiceFeeds System

Bsc-project 2008-2009 IN3405

- Danesh Harjani (1217488)
- Wai kon, Tse (1217704)

Delft University of Technology

EEMCS Faculty

Exam Committee

- Ruben van Einatten
- Bernard Sodoyer

Ver 0.12 17-03-09

Preface

In this report we will be talking about our 2 ½ month internship at "Greetinq". The internship is meant as a Bachelor-project at the TU-Delft and we were given the task of building a prototype system for them called 'VoiceFeeds'.

Summary

With our internship at Greetinq coming to an end, we would like to write our experiences while working on the project and with the guys from Greetinq. First of all, it was a very instructive period for both of us. Because we worked on a new project, we could have taken any direction we wanted with the project. This included its design to the frameworks used.

First, we made the required documents in order for us to begin the real work. These included the planning, requirements of the project and designs. Aided by Ruben, we quickly drew up some designs and made some preliminary research on the frameworks that were recommended. When all the documents and designs were in place, we began coding.

In the beginning, the coding was fast. Skeleton classes were auto-generated and the database was also generated. Then we began coding the back-end of the system without the use of frameworks. When all the basic building blocks were coded, we started using the frameworks. We started implementing the actions/behaviors and then problems started to show itself.

Problems arise when we didn't know how to do some things using a framework or the interoperability between frameworks. Some examples we encountered were implementing services in GWT. GWT services are really handy, because they can work with Java object in the back-end but it's very clumsy to work with. Another problem we encountered with GWT was working with non GWT services. The smpp framework was very hard to work with, because it's largely badly documented and the working examples were out of date.

And finally, we are happy with the result we've achieved in these few months working at Greetinq. It was a pleasant experience working with the guys from Greetinq and we give our thanks for giving us a chance to work with them.

Contents

1. Introduction

In this document we will be discussing the entirety of the 'VoiceFeeds' project, developed for out Bachelors Project at the TUDelft. We will begin with an introduction of the system to explain what the system is, what it does and how it works. We will then continue with the design of the system, from initial design on paper and UML to the chosen frameworks and their pros and cons. We will then discuss the implementation of the system, the design decisions made in the early and late programming stages and the special issues that needed to be solved. We will conclude on the results of the project, recommendation for further work and reflect on the time we have spent on the project.

2. Preliminaries

2.1 Introduction to 'VoiceFeeds' System

The Voicefeeds system has already been described in the 'Plan van Aanpak' report, however we will also provide description of the system here.

The system is the prototype of a content management system used to manage and send mass-voicemails through 'Greetinq's' own voicemail system. The system is to be used by companies; represented in the system by Accounts; to broadcast VoiceMessages via VoiceMail to mobile phone customers. By creating Channels; which are containers for VoiceMessage content; and allowing mobile phone customers (Subscribers) to subscribe to these Channels via a user-interface that can be imbedded on their website, they can then offer their customers a new service. 'Greetinq' can then generate revenue from these companies based on the amount of messages sent as well as other criteria.

The system also contains a System Admin Interface allowing the main admin at 'Greetinq' to manipulate users and accounts.

3. Design and implementation

This section deals with the design decisions made during the course of the project. We will begin with the decisions made in the early (planning) phase and continue with the decisions made on which frameworks to develop our project on and discuss the pro's and con's of these frameworks.

We will then continue with the changes and updates to the design made during the implementation phase. We will discuss changes made during the early programming phase as well as the late programming phase, and discuss special issues that arose during development as well as the solutions we implemented.

3.1 Design

3.1.1 Design decisions made in early phase

Entities/Actors

We recognize 4 actors in the system:

1. Subscriber

   This is the actor that will be subscribing to feeds in our system and thus be the recipient of the voicemail messages

2. Channel Administrator

   This is the actor that manages his companies account. He has complete access to all CRUD-functions available to channels, content, content admins, subscribers, account information and balance information in the system.

3. Content Administrator

   This is the actor that manages a specific set of channels belonging to an account defined by the permissions given to him by the channels Channel Administrator. He only has access to CRUD-functions on content.

4. System Administrator

   This is the actor that manages the users and accounts in the system. He has access to all CRUD-function pertaining to users and accounts, but does not have access to specific channels.

Models

During initial design we recognized the following models in the system:

- Account
  The representation of a company account in the system
- Cast
  The representation of a channel with a 1-time message broadcast
- Channel
  The representation of a channel in the system
- ContentAdmin
  The representation of a Content Admin in the system
- Delivery
  The representation of the delivery of a VoiceMessage in the system
- Feed
  The representation of a channel able to broadcast multiple messages
- FeedAdmin
  The representation of a Channel Admin in the system
- Message
  The representation of a VoiceMessage in the system
- Subscription
  The representation of a Subscription linking 1 subscriber to 1 channel
- SysAdmin
  The representation of a System Admin in the system
- User
  The superclass for ContentAdmin, FeedAdmin and SysAdmin

After initial programming we recognized one more model in the system:

- Permissions
  The representation of the permissions of a user in the system

Interactions with other systems

Our system interacts with 2 other systems:

The 'Greetinq' VoiceMail-push system:
This system is used to deliver our voicemail messages to the subscriber. We interact with it through a multi-part HTTP-post request sent to the server the VMS system is running on.

MOLLIE-SMS gateway:

This system is used to send SMS's to subscribers during the handshaking process for subscribing to a channel. We interact with this system through the SMTP-server provided by 'Greetinq'.

3.1.2 Decisions made while choosing frameworks

We have searched for frameworks that would be of use to us to ease the development of the project. In the 'Voorbereidend Onderzoek' report we detail the frameworks that were chosen in our initial design phase. However there are additional frameworks that were either chosen during development time or were considered but ultimately not used.

Considered Frameworks

Spring; a widely used MVC framework for building java applications; was considered but ultimately didn't get used. The reasons were two-fold, first we had already decided to use Google Widget Toolkit (GWT) and were not sure how well these 2 frameworks would interact, and secondly Spring is overloaded with functions that we did not necessarily need for our project. However, we were interested in the Inversion-Of-Control container available in Spring, but after looking at the available documentation of the I-O-C container implemented in Spring we found it to be very hard to use. Thus we decided against using any of Spring's functionalities.

Chosen frameworks

- GWT

  A java/javascript framework used for building websites in Java and then cross-compiling to JavaScript.

- Hibernate

  A Java-framework used for mapping Plain Old Java Objects (POJO's) to database tables based on XML.

Pros and Cons of frameworks

Pros

GWT

Java framework for building websites without the need to learn HTML or Javascript. GWT programming is just like programming a GUI in Java and then cross-compiling to Javascript. Implements its own version of standard AJAX functionalities such as Async RPC calls with java objects instead of XML; making development easier.

Hibernate

Java framework used map tables to java objects. Has its own SQL called HQL; SQL for Objects. Everything done in HQL is done on the object and not on tables and table properties. Very easy to get the POJO's working and mapped to the databases. The true strength lies in its flexibility. Adding functions to handle Use cases related to the database is simpler than using JDBC.

Cons

GWT

- Pre-defined directory structures
- Steep learning curve
- Doesn't look like a professional site with the provided styling. Widgets created are fairly barebones and ugly; CSS can be used to style widgets but requires a lot of development time if the developer is not already proficient with CSS styling of websites.

Hibernate

- Some functionalities require thorough reading of the documentation to understand how exactly the framework handles certain functions.

- Serializing of model mapped objects for use in RPC calls is sometimes not possible due to the attributes of the created Hibernate-Model objects.

3.2 Implementation

Before starting with programming, we used Jude as our main tool for UML modeling. Therefore it was easy to start implementation because of the auto-generated packages, classes and java code created by JUDE after exporting our UML diagrams.

For our database modeling, we used DBDesigner Fork. As the original DBDesigner is owned by MySql (Sun) and has been made into MySql Workbench and in our experience Workbench has been buggy and slow to work with, frequently crashing during normal runtime.

Because we chose MVC architecture, there are 3 main packages that form the main part of the final system.

3.2.1 Design decisions made in early programming phase

Structure

The system is built in 2 parts, the front-end and the back-end. Because the front-end was completely built with GWT we decided to split the project into 2 separate projects and then integrate them later. The Models and Controllers are being implemented in the backend and the Views are being implemented in the frontend. At the end of the project we integrated the 2 projects into 1 final WAR file, to be deployed on any web-server.The directory structure for our final integrated project is:

+---build	contains the compiled backend and frontend files
+---dist	contains the WAR file created after running the ANT build script
+---docs	contains Java-Docs as well as all project related text files
+---lib	contains external libraries used by the project
+---src	contains the source files for the project
+---tests	contains all the tests in the system
| +---backend	contains back-end tests
| +---frontend	contains front-end tests
+---backend	contains the source files for the back-end

+---frontend	contains the source files for the front-end
+---VoiceFeedsGui	Contains the source files for the GWT-project along with HTML / XML / CSS files and used images etc in the website

Naming

Because of special structure/package requirement from GWT all models and controllers are put in packages according to GWT naming conventions. Models are in package 'com.greetinq.voicefeeds.client.models' and controllers are put in 'com.greetinq.voicefeeds.server.controller' etc.

All the controller classes follow a naming convention in the form of [ModelName]+[Management]. For example a controller that manages the User model will be called UserManagement.

Use of GWT-RPC instead of XML-RPC

Although we were originally supposed to used XML-RPC for our remote procedure calls to the backend of the server we realized early on that GWT-RPC offered a much more integrated way of making these calls, by using java Objects instead of XML formatted data. This saved us a lot of time as adding and changing the functionality of a RPC was much easier.

Not using Test Driven Development in the front-end

Although we were supposed to develop both the front-end and back-end using the TDD model, where you would write a test first and then implement the function it became apparent that this was a waste of time in the front-end due to the fact that all the business logic for the program; the stuff that actually needed to be tested; was contained in the back-end. Therefore we decided to test the front-end with usability-testing instead of unit-tests.

3.2.2 Design decisions made in late programming phase

This is certainly not a surprise to us that we will be changing some specifications or a decision we've made earlier in the design phase. As we continue into the development phase, requirements became clearer and so did some requirement that that we didn't think of in the early phase of design.

What we did was update our requirements and update the related documents. What really helped was designing a flexible architecture that is easy to incorporate changes. This was also due to the facts that we are designing a system from scratch and not build upon an existing system. If it were an existing system, incorporating changes would be much harder. Also using a framework such as Hibernate was of help too, because if a model got changed, all it needed was an updated hibernate mapping and everything will work just fine.

In the last phases of development, we did a lot refactoring because the code got larger and we needed to manage the code better.

Implementation of the Permissions and PermissionsManagement classes
During development of our application we did not really consider the need for Permissions in the system, we knew they would be needed however we thought it to be easily implemented in the User class and would only be needed for certain functions such as whether a User had access to a Channel in the system. However after getting further into development and realizing that permissions were to be used throughout the whole system instead of just user verification we decided to make a Permissions and PermissionsManagement class to handle these. Details on how these work are to be found in the programming notes.

Use of ENUMS instead of INTEGERS to define the Type of an Object
During development of the system we did not really think about how the program would be developed on in the future by another team or even ourselves. Therefore when implementing objects that had a special State or Type we chose to use ints (0 = type1, 1= type2 etc.) instead of the more commonly used Enums. After this was pointed out to us by our project leader we decided that using Enums would be a better choice for our system. This however led to a lot of rewriting of any code that accessed these types, predominantly the tests.

3.2.3 Special issues needed to be solved

Model Objects created by Hibernate unserializable
During development we ran into a very big problem concerning objects created by Hibernate, any object that contained a reference to another hibernate object was unserializable and thus could not be passed from the server to the client without causing errors.

Solution: Creating special [Model]GWT objects for each unserializable object and creating an ObjectTranslator in the server that would convert a [Model] object to its [Model]GWT counterpart and vice-versa.

4. Results and Future works

4.1 Results

What we have here is a prototype of the VoiceFeeds system with the requirements implemented and a working frontend.

4.2 Future works

For people that are extending the 'VoiceFeeds' system, we recommend you read through the delivered documents and especially the Programming notes. The programming notes describe in detail how to extend the functionality of the system without causing a steep learning curve.

4.3 Conclusion and reflection

As for our conclusion, it was a very rich learning experience for us. We've learned a lot in these few months, especially when working with new frameworks and new development tool chains.

At the end of the project we did find ourselves going over the deadline, partly due to less time available due to conflicting school schedules and availability and also due to several problems we ran into during integration of the frontend and backend.

Some things that we would change:

- Integrate the primary builds exactly as you would integrate the last builds While working on our first 2 builds we decided to use an external JAR file as the backend of our system, imported into the GWT project of the frontend. While this worked out fine at the start when it comes the time to build the projects together into a WAR file for deployment several problems arose which we had not seen during the first builds. Therefore it would be easier to build a WAR from the first build onwards and not have bothered with external JARs.

Reflection:

Although GWT is an excellent package to rapidly prototype and develop a website, the amount of time to learn the package combined with the barebones look of a website created with GWT make it unappealing to

develop further projects in. The use of a Javascript library such as jQuery with the mass of tutorials available to it would produce a much better looking product in the end. Multi-broswer support however would be harder to handle using just a javascript approach, and thus we would have likely spent more time on the website.

A javascript library like Dojo or Scriptaculous coupled with JSON could also be an alternative to jQuery. This means that all the technologies are built on web-technologies and are open-source. Especially for JSON, it would make it simpler for us to pass objects around especially when going from JAVA – Javascript or the other way around.

The use of the Hibernate framework worked out very well for us, programming with Hibernate is simple and allows us to rapidly build our database models and link them to our controllers. We would use Hibernate again in another project with a database

5. Appendix

All the documents that belong in this appendix can be found together with this report. The needed documents were not included in this appendix because the documents are long and will make the document overloaded with information.

List of document and their file names:

1. Requirements Analysis Document – Requirements Analysis Documents.pdf
2. Architecture Design Document - Architecture Design Document.pdf
3. Technical Design Document – TDD.doc
4. Plan van aanpak – Greeting voicefeeds - Plan van aanpak.docx
5. Voorbereidend Onderzoek - Voorbereidend_Onderzoek.docx