Communicating with a Large Customer Base via Mobile Messaging

Richard Hable
Evolaris Next Level GmbH
Hugo-Wolf-Gasse 8-8A
A-8010 Graz
+43(0)664/8414-439
richard.hable@evolaris.net

Thomas Ebner
Evolaris Next Level GmbH
Hugo-Wolf-Gasse 8-8A
A-8010 Graz
+43(0)664/8414-434
thomas.ebner@evolaris.net

ABSTRACT
The widespread use of mobile phones has opened up new ways of communicating with large numbers of people. Companies can use personalized SMS messages as a fast and cost-efficient means of providing customers with information and receiving immediate feedback. In this paper the design, development, and use of a mobile marketing platform is described both from a technical and a user-experience point of view. Solutions are presented which allow fast and reliable sending of large numbers of messages. A central event processing mechanism has been implemented which reacts to incoming messages and processes notifications received from external systems. Timer-based services perform regular tasks like sending informative messages to customers and business users of the platform. Customer data is managed with automated registration and deregistration, data exchange with external systems, selection of destination groups based on user attributes, and the collection of a comprehensive history of customer interactions. Intelligent supervising tasks which automatically recognize exceptional conditions like unusually large message quantities within a specified time frame are performed in the background. An administrative web interface of the platform with comprehensive message monitoring functions allows taking appropriate action in such cases. After describing this and other unique features, experience from development and practical applications of the platform is presented which allows evaluating the project and summarizing the lessons learned during five years of development and deployment of the platform.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – *domain engineering*

General Terms
Performance, Design, Reliability, Security.

Keywords
Middleware, Messaging, Mobile communication, SMS

COMSWARE 2011, July 04-07, Verona, Italy

DOI 10.4108/comsware.2011.3

platform grew to support more advanced content distributed via multimedia messages (MMS) and e-mail. Enhanced messages could also be used as a starting point for interaction with customers based on different technology and media, for example containing links to Web sites and multimedia content within WAP push messages or containing tags (2D codes) usable for couponing services.

A characteristic trait of such services is that no special software is required on the client side–just about every mobile phone could be used for mobile marketing services without requiring additional configuration, installation or technical knowledge from the average customer. This turned out to be a big advantage especially during a time when customization of mobile phones was limited mainly to downloading ring tones and possibly some simple Java games. Large numbers of customers could thus be reached with no more information required than their mobile phone numbers. Despite the technical limitations, sending personalized messages to mobile devices turned out to be a very effective means of gaining the attention of customers with low cost compared to other communication means like advertising mail.

In addition to the technical tasks of sending and receiving large numbers of personalized messages of different kinds, the server platform was also needed to support campaigns based on these messages. It was necessary to manage customer information and to provide administrative user interfaces to clients using the platform to communicate with their customers. The large number of customers reachable via mobile messages posed challenges concerning the automation and monitoring of interaction flow between the platform and the customers. Customers sending and receiving messages and using other services supported by the platform had to be served independently of time and location without requiring manual intervention. Thus, it was necessary to automatically react to incoming messages and requests from external systems with actions appropriate to the currently active campaigns. Also, timer services had to be implemented which allowed performing tasks automatically which had to be done repeatedly or time-delayed. Since different clients required different interactions with their customers, a flexible modeling and scripting system was developed and successively extended to allow configuring new campaigns without having to reprogram and redeploy the platform each time new demands were made on it.

The following chapters will describe both technical and functional solutions based on the initial and ever-growing requirements, and the technical foundations used for implementation. Experiences gained with practical use of the platform and a critical analysis from a development view will allow an evaluation of the decisions made and hopefully provide insight into the success factors and

things to consider when developing similar middleware systems today. This also means that changes in technology, available frameworks and also the changed mobile device dissemination today compared to five years ago have to be considered for these conclusions to be of any value.

2. FEATURES

The first tasks required from the platform were to send SMS messages and react to incoming SMS messages in an appropriate way. Clients had to be able to use the platform to send the same message to many customers at once ("group SMS service"). Customers on the other hand had to be able to send messages to a predefined mobile telephone number (MSISDN) and receive a response message containing some information ("information SMS service"). Such features would also have been available via the web interfaces of commercial SMS gateway providers. However, in order to achieve a highly integrated system, it was decided to implement a custom enterprise application, which relied on external systems only for actual message transport.

2.1 User Management and Security

Since the platform was to be hosted on a central server as a multitenant application, all clients using the platform for message sending and all customers sending messages to predefined phone numbers had to be managed within a single platform database.

It was important to distinguish between different business clients, because a client of one company may never access customers of other companies. Also, different business clients were allowed to use different services within the platform. Since all business users accessed the same instance of the platform, a unique user name had to be assigned to each of them and they had to login on a platform web interface with a password in order to correctly permit or refuse access to appropriate parts of the platform functions and its data. Since the number of users belonging to a single client organization could become very large, an additional grouping mechanism was implemented. Users can be explicitly added to and removed from so-called user sets in order to group them for later tasks like sending them specific messages. Later, a dynamic extension of this concept was implemented for automatic selection of users according to attributes like their age or place of residence. Since many business clients already had a customer base which they wanted to use together with platform services, import and export mechanisms were required which allowed exchanging information about users both explicitly in the platform user interface and automatically using web services and file transfer.

Within the platform database, users can be assigned different roles. Depending on these roles, they receive different rights for using platform functionality. A platform user may for example have the role "customer consultant" which is configured to give the right to access a user data editing page and a message sending form. In order to clearly define the amount of data users are allowed to access, they are also assigned different administrator status levels. The lowest level only allows accessing personal data, higher levels allow accessing data of other users or clients, and the highest level even allows changing configuration data vital to the correct functioning of the platform. In order to prohibit proliferation of data access rights and minimize possible damage by intruders, platform users may only assign administrator status levels below their own level to other users. Thus, the highest administrator status cannot be assigned via the web interface at all, but must be configured by a system administrator directly within the database.

For a highly security critical banking application, the built-in security features of the platform have been complemented by installing a separate instance of the platform and its database on a highly secured server accessible to only a specific set of clients.

2.2 Message Sending

SMS and other mobile messages have to be sent via a mobile gateway provider. This is a commercial service which can be accessed via a web interface to initiate sending a message to an arbitrary MSISDN independently of the service provider used by the customer. Connections to different gateway providers with different message types and billing models have been implemented to allow maximum flexibility according to the business case.

In addition to SMS, EMS and MMS messages sent via gateways, later also e-mail message sending was implemented, both for administrative messages to business clients and for messages to customers with advanced mobile phones like the iPhone which use e-mail as a preferred means to send and receive messages with rich multimedia content. E-mail messages are sent using standard mail server software installed on the platform server and thus do not lead to additional costs.

The platform serves as a unifier for different types of messages and different means of message transport. All the business user has to do is to select the appropriate destination MSISDN or e-mail address and compose the message either using simple text or enhanced content with multimedia attachments. The platform then chooses the appropriate means of message delivery.

Both single messages and large quantities of messages are sent by the platform. In the latter case a flexible placeholder mechanism can be used to personalize messages, for example by automatically creating personal salutation text based on customer names stored in the platform database.

Sending a large number of messages may take a lot of time. In order to avoid making the platform user wait in front of the web interface until sending is done, the platform stores message tasks, i.e. one or more messages with similar content, for later delivery. A different page in the platform web interface allows checking which messages already have been sent and which messages are still scheduled for sending.

Time-delayed sending of messages is also supported, thus allowing the business clients to select an appropriate time for contacting their customers independently of message composition.

2.3 Incoming Messages

Phone numbers and e-mail addresses used for message sending can also be used to receive incoming messages. It is important to react to these messages as they are the simplest way for customers to provide feedback. In order to allow automatic classification of incoming messages, both source and destination phone numbers and e-mail addresses are evaluated. The destination phone number or e-mail address can be used to determine which service a customer responded to. The source phone number or e-mail address can be used to unambiguously identify the customer within the group of users managed by a business client.

Since only a limited number of phone numbers and e-mail addresses is available, a mechanism has been devised which allows additional classification of incoming messages according to keywords within the message text. This way different services can be provided via a single destination address, possibly even shared between different business clients. Additional features of incoming message processing include automatic content analysis

and taking over parts of the message content as user attributes. This way, users can, for example, send their names and birth dates in an SMS message in order to register for a service.

2.4 Content

The platform has to store multimedia content in order to automatically serve users requesting artifacts like pictures and ring tones. Since different mobile devices require different multimedia file formats and content quality according to their hardware capabilities, different versions of the same content can be stored which are automatically selected according to the devices accessing the platform based on a WURFL configuration file [1]. This way, for example, customers with different mobile phone types requesting the same video film may receive the film in different encoding format, different resolution and thus different quality appropriate to their devices.

2.5 Offers

In order to get customers interested in mobile services, clients of the platform may propose offers for those registering for a service. The platform has to keep track of the number of offers redeemed by a single customer or in total and react differently based on additional information available about the current user. An advanced management system has been implemented for offers distributed via tagging (2D codes). Customers receive tags via specially encoded EMS messages, which unambiguously identify them at the platform. They may then put their mobile phone with the displayed tag on a scanner which prints gift coupons that can be exchanged for diverse services. Within the platform user interface, a special graphical editor can be used to design the appearance of the gift coupons and to select customers eligible to the services. Each time a customer tries to retrieve a gift coupon at a scanner, the platform is contacted via the wireless GPRS network and uses the configured information to decide which coupon, if any, shall be printed.

2.6 Interaction Modeling

In the beginning, the platform was used for simple services like sending mobile messages to groups of customers and responding to requests for information received via mobile messages. Later, more and more advanced services were developed, which required increasingly complicated automatic interactions between customers and the platform. Thus, an interaction modeling system was developed, which allowed defining the tasks to be performed whenever incoming messages were received by the platform. In order to allow business clients to define and adapt interaction behavior themselves, this system was provided with a simple and familiar user interface, which hides the often complicated internal programmed operations. Many different strategies of analyzing incoming message content are supported. Actions like adding customers to user sets, which normally are performed manually within the platform user interface, can be modeled to be automatically performed each time a message is received. The modeling system also provides features like conditional execution (see Figure 1) and subroutines without requiring the user to learn the syntax and concepts of a real programming language.

The interaction modeling system is also used to automate tasks independently of interaction with customers. Timer-based tasks can perform delayed and repeated tasks, and the platform can react to requests from external systems using the same mechanism as for incoming message processing. With its interaction modeling capabilities the platform has thus evolved into a flexible event-based system for automated communication with business clients, customers, and arbitrary external systems connected via the Internet.

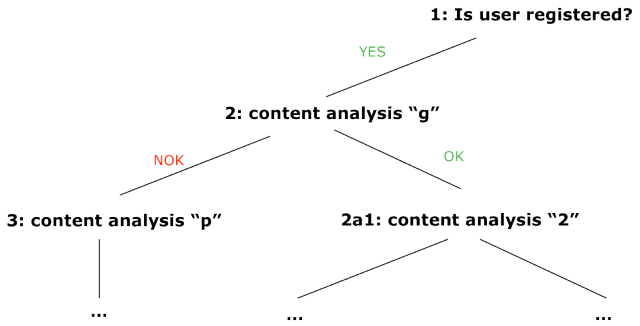

Figure 1. Interaction Control Flow

2.7 Reporting

The platform automatically communicates with a large number of customers, causing both significant messaging costs and public attention. Therefore, it is vital to regularly and attentively monitor the actions performed by the platform and the reactions of the customers.

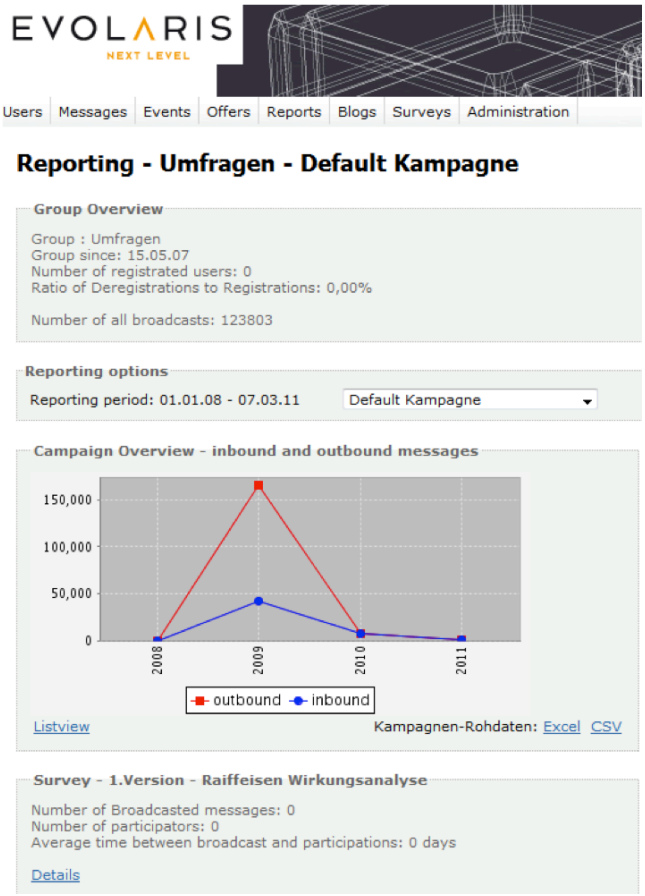

Figure 2. Standard Report

Both automatic and explicitly modeled logging has been implemented, which can be used to create reports and statistics to be evaluated by users of the platform. Based on the number of registrations, deregistrations and active contacts initiated by customers, the success of marketing campaigns can be measured. Different response rates for different service offers can be taken as indicators of customer acceptance and used to optimize subsequent campaigns.

In order to be able to react immediately to exceptional conditions, thresholds can be defined for different classification numbers to automatically send notifications to platform users. Thus, beyond ensuring correct technical operation of the platform, it is also possible to automatically check for reasonable accomplishment of business goals.

Within the platform web interface both standard reports for recurring platform applications (see Figure 2) and reports specifically designed for individual client projects can be retrieved. In addition to that, interfaces to the platform have been implemented, which allow remote access to statistics calculated by the platform. Mobile applications for modern smart phones like the iPhone have been created which display this information in a way appropriate for monitoring of the platform.

2.8 Mobile Application Support

Beyond services based on message sending and receiving, the platform has also been extended for applications which use different ways of interaction with customers. Nevertheless, the integrated user management of the platform still remained a vital component of administrating customers using these applications. Also, mobile messaging is still important at least in order to establish initial contact with customers.

2.8.1 Mobile Web Applications

Prolonged interaction with customers via standard mobile messages like SMS can become expensive and cumbersome for both business users of the platform and the customers themselves. Therefore, with the advent of more and more capable web browsers within mobile phones, it was decided to take advantage of the advanced formatting and interaction possibilities of mobile web sites. However, hardware and software capabilities of mobile phones are differing to a large extent. Therefore, mobile web sites are created using a special platform independent markup language. Pages encoded this way are automatically converted to HTML or WAP markup code taking advantage of the screen resolution, processing power and available browser features of the specific device using the application.

Both web applications created specifically for single customers and generally usable applications were created and hosted on the platform server.

The most frequently used general web application is a mobile survey tool (see Figure 3). This application allows customers to take part in surveys using their mobile phones whenever and wherever they want to. Survey application flow with different types of questions can be configured within the platform web interfaces. The platform messaging system is used to send invitations to customers to take part in a mobile survey. Messages containing links to the mobile survey web application are sent to selected customers at appropriate points in time. Optional parameters within these links can be used to identify users taking part in a survey. Each time customers open such a link, web pages based on the configured survey application flow are displayed on the mobile phone and answers given in appropriate input fields are sent back to the platform. The platform collects these answers and provides extensive statistics about the responses. Also, export interfaces have been created which allow evaluating the results with external applications like advanced statistics tools.

Figure 3. Mobile Survey Web Application

2.8.2 Mobile Native Applications

Server support is also often required for native applications on mobile phones. In this case the platform provides services usually based on the HTTP protocol and using simple data encoding like the JSON format.

As prototype for a generally usable native application using the services of the platform a mobile customer card application has been developed (see Figure 3). This is a client application available for the iPhone and for smart phones based on the Android operating system which intends to replace the collection of loyalty cards stored in the wallets of most customers.

This application displays offers available to owners of loyalty cards on the screen of the mobile phone. The platform is contacted each time the application is launched and responds with a list of offers available for the user which can then be displayed and used at the checkout counter of the store taking part in the loyalty card system. These offers are configured within the platform web interface and stored in the platform database.

As an additional feature, a messaging system within the mobile applications has been implemented which can be used as a cheaper and more direct way to communicate with a customer than conventional SMS messages. For this purpose, the platform allows composing and storing messages to individual users of the customer card application. Each time the application accesses the platform, new messages are retrieved and displayed.

3. TECHNICAL IMPLEMENTATION

Platform development started in 2006 with initial requirements clearly indicating that a comprehensive server application managing persistent data would be needed. Technology decisions were made based on standards and established tools prevailing at that time.

3.1 Base Technology

The common choice for professional server applications five years ago and to a large extent still today is to use Java technology. This allows operating system independent server application development and deployment.

Java was therefore also chosen as the programming language to use for platform development. Java programs are compiled to system independent object code (class files and archives) which can be executed within a virtual machine available for all relevant operating systems.

In recent years, other programming languages have become available which can be executed on Java virtual machines. This allows using the same libraries as Java programs and interoperating with existing Java code. The platform takes advantage of this by integrating the scripting language Groovy [2] for tasks which require concise interpreted code which can be easily exchanged at runtime. Interaction modeling uses Groovy scripts for flexible interaction configuration within the platform web interface without extension and redeployment of the platform. Expressions written in Groovy can be used during mobile survey configuration to specify conditions evaluated during survey processing.

The operating system independence of Java also allows developers to use Windows and Macintosh-OS based personal computers while the created software is deployed on Linux server systems.

3.2 Middleware

There is a big difference between stand-alone programs intended for single users and server software allowing external access and supporting multiple users at the same time. Different standards and tools are available which support creating server applications of this type, most commonly used among them are web servers which handle HTTP requests from web users and web services.

3.2.1 Application Server

The Java EE (Java Enterprise Edition) standard [3] is a vendor independent specification for server applications defined in the Java Community Process. This standard adds an additional layer between the Java virtual machine and application specific software. Applications are created as individual components which can be deployed on an application server. This is a software container providing services like transactions, security and concurrency.

The platform was initially developed to be deployed on the open source JBoss [4] application server. This application server contains an integrated web server, Apache Tomcat [5], used for the platform web interface and web services provided by the platform. Another application server feature initially used by the platform was the integrated message queue support, which allowed asynchronous message sending and receiving. The Java EE standard to some extent makes sure that application servers of different vendors and projects remain compatible. Thus, software developed for a specific application server can also be used within other application servers. Nevertheless, differences in configuration and built-in components make this a non-trivial task.

Using a Java EE application server leads to disadvantages mainly during the development process: development turnaround time, i.e. the time required for an edit-compile-deploy-test cycle, is increased by high resource requirements and slow deployment of single components. Also, extensive application configuration requirements can lead to problems which are difficult to solve during development and deployment. Since large parts of the platform only required the presence of a web server, development therefore could often be performed using only the Apache Tomcat web server software. Nevertheless, having to maintain different configurations for deployment within the web server and within the application server lead to additional configuration effort and error-proneness. In the end, the features provided by an application server in addition to a stand-alone web server were considered to be not important enough to accept the disadvantages during development and deployment. Therefore, three years ago the application server features used by the platform were replaced by manually programmed management code and the platform has only been deployed on stand-alone Apache Tomcat web servers since then.

3.2.2 Database

Concurrent access to shared data requires transactional persistent data storage. Relational databases were considered the standard way to deal with such requirements five years ago. Alternative approaches like object-oriented databases were not established in industry, although they would in principle have been a better match for software created with object-oriented programming languages like Java. Today, object-oriented databases still play an insignificant role in industrial software development. However, databases using non-relational data structures, so-called NoSQL databases, are getting popular for high performance and large volume applications now.

The platform has been using a single relational database instance with a single database schema from the beginning. As expected, run-time performance of the platform is mainly limited by the speed of operations on this database. However, using approaches like explicitly provided table indices, caching in memory and blocked data writing, performance could be improved sufficiently each time performance became an issue. Nevertheless, very large data volumes required, for example, by video films served by the platform, are stored in the file system of the server instead of the platform. This is possible because this kind of data is usually uploaded once to specific server directories and is not changed afterwards. Therefore, transaction support and advanced query mechanisms provided by a relational database are not needed.

Relational databases can be accessed via the low-level JDBC interface defined in the Java EE standard. However, in order to alleviate the conceptional differences between object-oriented programming and relational database access, also known as “impedance mismatch”, the object-relational mapping tool Hibernate [6] is used in the platform to access database structures with standard Java objects. This tool also hides subtle differences between different database management systems (DBMS). This way, although the platform has been almost exclusively used with the open source DBMS MySQL [7] from the beginning, systems of other vendors could also be used without having to adapt the platform software. In order to verify this possibility, the platform has been successfully tested with an Oracle DBMS.

3.2.3 Web framework

The Java EE standard defines ways of creating and deploying dynamic web pages. These are web pages containing not only static texts and multimedia content, but also content dynamically created from data repositories and processed by business logic. This is required for web applications, which allow displaying and modifying data from a database and provide monitoring functionality for a permanently running system. The user interface

of web applications usually is a compromise between the speed and comfort of native applications and the technical restrictions imposed by the stateless HTTP protocol and the limited interactivity supported by the HTML standard.

Servlets provide the most basic mechanism for dynamically created HTML pages served via web servers. These are Java classes which, after having been installed on a web server within web archives (WAR files), are invoked for incoming HTTP requests and create responses based on request parameters and static session variables. In order to simplify creating HTML pages, the Java EE standard also supports Java server pages (JSPs), which allow specifying HTML tags mixed with Java code, additional tags and expressions defined in a higher-level expression language. This way, pages served to the client can be developed more closely to the desired result than with pure Java code.

Nevertheless, creating interactive user interfaces based on HTML standard, which were initially only intended to serve documents containing hyperlinks, is difficult and laborious. Therefore, many different frameworks have been created which support both the creation of HTML pages with advanced user interface elements and improved page flow. The Java EE standard also defines such an improvement based on JSPs: Java Server Faces (JSFs). During the conception of the Evolaris platform, however, mature implementations of this standard had yet to come. Instead, the most frequently used web framework was Struts [8]. This framework supports centrally configured page flow and a tag library supporting form input and validation. The platform uses Struts for all web user interface components in order to preserve the consistent look and feel of the user interface and to allow seamless integration of new components.

3.3 Development Tools

For software developed by a team of several developers during several years, it is important to settle on consistently used development tools in order to guarantee continuous development progress and reliable release management.

3.3.1 Integrated Development Environment (IDE)

The most frequently used Java IDE is the open source framework Eclipse [9]. Diverse packages and additionally installable plug-ins are available which add support for enterprise application development and even other programming languages and target systems not based on the Java virtual machine. Therefore, developing the Evolaris platform within Eclipse IDE for Java EE developers was a logical choice. This allows creating, configuring, and deploying software (web archives) on Java application servers and web servers in a single consistent tool environment.

3.3.2 Source Code Management

In order to easily track changes to the platform and to be able to restore older versions if necessary, all source code is stored in a repository. The open source software versioning and revision control system Subversion has proven to be adequate for this task and is now used for all software artifacts created by Evolaris. Plug-ins for the Eclipse IDE allow easy source code management within the familiar development environment.

3.3.3 Building and Testing

Developers build software within an IDE and test new functionality according to the changes they made. However, each time a new version of the platform has to be deployed on a production system, a new complete version has to be built and tested. This means that even parts of the platform which have not been changed recently have to be built and tested anew. In order to automate this process as much as possible, build and test scripts have been created. These scripts can not only be performed when versions are due, but also have been used for regular nightly builds and regression tests. This provides early feedback for developers about unintended changes and errors introduced when modifying the platform or implementing new functionality.

A collection of SQL and Ant [10] build scripts has been created which supports automatic creation and initialization of database tables, automatic creation of Hibernate data access objects, and packaging of multiple JAR and WAR files at once. Recently these scripts have been converted to Gant [11]. This allows using the scripting language Groovy within build scripts for additional flexibility.

For nightly builds and tests the Jenkins (former Hudson) [12] Continuous Integration tool is used. The current version of the platform source code is automatically checked out, built and deployed, and automatic tests are performed. Unit tests based on the JUnit library and web user interface tests using the Tellurium [13] automated testing framework have been created for these tasks. Results and statistics of these nightly activities are automatically provided on a web page, and notifications about failures are sent to the developers via e-mail.

3.4 System Architecture

3.4.1 Design Decisions

The platform had to support many users in parallel. Some of these users had to be able to actively use functionality provided by the platform via their personal computers, others would only communicate with it via mobile devices. In order to make these different kinds of access as easy as possible, it was decided that users should not have to install additional programs on their devices. Built-in and pre-installed software like web browsers and mobile messaging applications had to be sufficient.

Therefore, the platform administrative user interface had to be implemented as a client-server web application. All application layers, from low-level database access to HTML page composition, had to be implemented on the server. Complementary technology like JavaScript and CSS (Cascading Style Sheets) nevertheless allowed a comfortable user experience when accessing the platform with a standard web browser.

Communication with mobile devices was more restricted. Five years ago, hardly any mobile device supported web browsing sufficiently for web applications. Therefore, in the beginning all communication with mobile devices was restricted to SMS, MMS and e-mail messages. Later, simple WAP and HTML pages optimized for mobile devices were added.

3.4.2 Modularity

Modularity is a crucial factor when creating large software systems which have to be maintained and extended for years. The Java programming language itself provides little support for modular programming: only within class files, the base structuring building block of Java applications, a clear separation between private and public elements can be enforced. Logically associated class files can be combined to Java archives (JAR files) and similar archives used for deployment in web servers (WAR files) and application servers (EAR files). However, no distinction is made between classes required only within an archive and classes intended to be used by external code. Different solutions have been proposed to support structuring of Java applications on this

level; however, they require additional middleware or regular application of tools enforcing structuring according to a configured architecture.

The platform instead takes advantage of the modularity gained by the project concept of the Eclipse IDE. Different types of projects have been created, each of them both technically and logically confined to a specific task. Interdependences between these projects are limited to non-cyclic references, thus ensuring the basic requirement for independent development and testing.

3.4.3 Horizontal Layers

The platform software is separated into three layers according to the standard three-tier client-server architecture (see Figure 4). The presentation layer containing web user interface components belongs to the highest level. On the next level there is a business layer containing reusable processing components. The data layer implemented using the Hibernate O/R mapping system belongs to the lowest level. In order to enforce cycle free dependencies, code may only access components within its own layer or below. User interface and business components are therefore developed within separate Eclipse projects with unidirectional dependencies.

3.4.4 Vertical Layers

The complexity of the platform and its optional components require additional structuring according to feature domains. Therefore, within the single layers, separate Eclipse projects are used to separate, for example, code needed for the implementation of messaging from code needed for couponing support. These projects, too, are part of a cycle-free dependency hierarchy.

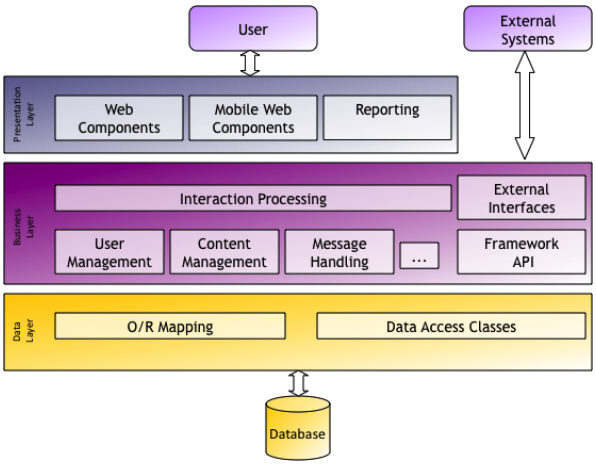

Figure 4. Platform Architecture

4. DEVELOPMENT EXPERIENCES

Platform development started approximately five years ago. A first version of the software supporting simple tasks like managing users and sending and receiving SMS messages was available and used in practice within a few months. Since then, the platform has been constantly improved and extended according to new requirements and application domains.

4.1 Base Development

After the principal technical decisions had been made, two developers started defining the initial data structures and creating the first administrative web pages. Later, up to four developers and one tester worked together.

Nevertheless, the initially implemented modules and data structures still play a central role for the platform and project-specific extensions.

4.1.1 Database

The platform requires a relational database management system in order to provide multiple services in parallel and to persist data across server reboots and software updates.

4.1.1.1 Modularity

All central database tables and their Hibernate mapping files and data access classes are stored within a single Eclipse project. This is necessary because in order to avoid redundancy and to enforce referential integrity, the database tables extensively refer to one another. This leads to cyclic dependencies which must not cross project boundaries. With the continuous extension of the platform, the number of database tables has grown to more than 100. In order to maintain overview, naming conventions are used to group tables according to their logical function even though they are part of a single database schema and defined within a single Eclipse project.

4.1.1.2 Schema Evolution

Data structures defined in the database schema are the basis of most higher-level modules of the platform and thus strongly influence the modular structure of all software artifacts. Since they store persistent data which has to remain valid across successive versions of the platform, they are also very difficult to change. Therefore, it is important to get database structures right the first time as often as possible. Strict development guidelines have therefore been defined which have to be obeyed when changing or extending the database. Developers are also obligated to keep database changes compatible with earlier versions and to send change proposals to their colleagues by e-mail before checking in any changes to the database schema. Nevertheless, refactoring and incremental extensions have made some database tables and individual table attributes obsolete. These tables and attributes are now clearly marked deprecated and may not be used within source code anymore.

4.1.1.3 Transaction Handling

As soon as the platform was used to support many customers at the same time, problems with inconsistent data and spurious dead-locks arose. First, a very radical, but also save method of "pessimistic locking" was applied. It ensured that access to the database was limited to one process at a time. Since most services of the platform rely on the database, this meant that services could only be performed sequentially. During the first applications of the platform, this mechanism worked rather well, since performance was not critical and most existing services could be easily implemented within short transactions. Later, however, services taking long processing time were required, and delays were perceptible in the platform administrative user interface. Also, during times of higher platform load, some services were automatically cancelled by the application server because timeout limits were exceeded. Finally, pessimistic locking was abandoned and tables are now only locked when absolutely necessary. Deadlock conditions are avoided by defining a strict non-cyclic locking order when locking more than one table within a transaction. Despite this optimization, timeout conditions still occur occasionally during times of high platform load. In order to minimize this problem, it has been taken care to avoid short-time accumulation of time consuming tasks.

4.1.2 User Management

Just about every application of the platform relies on the user management. Consequently, this has also been the part which required most extensions. Many attributes have been added to the central user table in order to be able to store all information accumulated in diverse projects. This also always required adapting and extending the platform user interface and other user managing components like user import and export. Finally, a generic means of adding application-specific attributes by configuration was implemented. Since then, most customer projects can be carried out without extending the user management parts of the platform.

Another challenge was the growing number of customers that had to be managed by the platform. For a project supporting communication with a large regional bank, more than 100,000 individual users are stored in the platform database. In other projects survey invitations are sent to a large number of users, whose names and phone numbers are imported from external databases. This way, new users are constantly added to the database. Since the platform stores comprehensive usage and logging information for each user, all user entries also have to be retained in the database during the complete duration of a project.

Having to handle very large numbers of users can become both a usability and a performance problem. In order to allow platform users to easily find and select customers, extensive filtering support had to be implemented, which allows finding users according to personal attributes like age and gender. In order to be able to easily divide large groups of users into manageable chunks, the concept of user sets, that is explicit assignment of users to arbitrary sub groups, was extended with a dynamic selection based on personal attribute. In order to efficiently select user sets in the database and thus avoid performance problems, the built-in table mapping support of the Hibernate library had to be partly replaced by explicitly invoked optimized SQL statements.

4.1.3 Message Handling

Another module which was part of the platform from the very beginning is messaging support. Initially, this only meant sending and receiving a small number of simple SMS messages. Later, messaging was extended to support other types of messages like MMS and e-mail and to allow advanced personalization via placeholders. With the growing number of messages sent and received by the platform, support for monitoring and intervention of message sending was needed. While the platform was using message queue support of the JBoss application server, messages scheduled to be sent could only be revoked by manual intervention of a system administrator. After the messaging part was rewritten for use on a web server, scheduled messages were stored within the platform database and could thus be manipulated via the platform user interface up until the time of actual delivery to an SMS gateway provider. The large number of sent messages required aggressive performance improvements like delivering up to 50 messages within a single database transaction. Data structures had to be partly de-normalized in order to efficiently write and retrieve the results of these bulk operations.

4.2 Client Projects

The platform is based on universally usable modules which, once implemented, usually stay in the platform to be available in arbitrary projects. However, many different client projects have been carried out which required specific implementations only during a limited time period. It would not have made sense to put everything into the platform and thus make it difficult to maintain for features only used once. Therefore, the platform was designed to be extensible with optional components. Technically, this is possible using Java archives deployed separately on an application server or on a web server.

4.2.1 Common Functionality

Since client projects rely on the platform basis, they have to be able to access data structures and functionality of the platform base modules. This is accomplished by adding Java classes of the platform base to the client project web archives. Client project services thus access the same database tables and call the same utility classes as platform services. In order to ensure reliable operation of client project implementations across different platform versions, client projects may access platform functionality only via a collection of Java interface classes which remain compatible even when the platform implementation is changed or extended.

4.2.2 Database Extensions

Client projects may define their own database tables in order to store project specific data. These tables may contain references to platform tables; however, no references from platform tables to project-specific tables are allowed. An explicitly evaluated configuration file with references to additional Hibernate mapping files is included within client project code to allow automatic retrieval of these additional tables by the Hibernate database interface library.

4.2.3 User Interface Adjustments

User interface web pages specific to client projects can be added to the platform user interface via a configurable menu system. Depending on the client project logged-in users belong to, different menu configurations are read from the database with links to appropriate platform and client project pages. Initially, the menu structure was stored within several tables, using relational references between menu bars, menus and menu items. This, however, turned out to be difficult to handle during deployment and version updates. Therefore, later a simple XML structure was defined which allows storing complete menu bars including contained menus and menu items within single XML documents. These documents are stored in a text field within a single table among other client project configuration information.

5. EVALUATION

Designing and implementing a platform during several years with continuously growing requirements and application domains has been both a challenging and rewarding task. Today, both efficiency of development and quality of the resulting product can be evaluated based on current technical know-how and the history of practical applications of the platform so far.

5.1 Reliability

As can be expected with any software system developed and extended during several years, correctness and stability of the platform have been a serious challenge. Extensive changes and improvements often endangered even basic features of the platform. It was necessary to enforce a more and more strict deployment process with defined testing phases both before and after installing new versions. Nevertheless, unpredictable down times and erroneous functioning during times of version upgrades remained a problem.

However, with the platform becoming more and more flexible, especially due to the inclusion of scripting support and

configurable user data fields, the need for platform extensions and deployment of new versions decreased. The reliability and availability of the platform accordingly became better and better. During the year 2010, for example, only five new versions of the platform had to be installed, with server down times between 10 and 40 minutes. Long down times were mainly caused by changes in the database schema which required the conversion of large amounts of data.

5.2 Performance

The performance of the platform is limited mainly by database access and communication with external systems.

When necessary, it has always been possible to sufficiently improve database performance using optimized SQL queries, blocked data access, and even helper tables with redundant accumulated data. Thus, the initial decision to rely on a single database instance hosted on the same server as the platform proved to be adequate.

The main limitation of platform performance caused by external systems occurs when sending large numbers of personalized SMS messages. Each message has to be transferred within an own HTTP request to an SMS gateway provider. Nevertheless, due to optimized database handling (50 messages at a time) and a high-speed Internet connection, it is still possible to transfer up to 10 messages per second. The largest message-sending task until now has been to send personalized SMS messages to more than 17,000 customers at once. Transferring these messages to the gateway provider took approximately 30 minutes and thus still allowed sufficient control over the time frame of message delivery to the customers.

5.3 Technical Evolution

The principal design and implementation decisions have been made more than five years ago, and therefore it makes sense to take a look at the base technology used by the platform and the current state-of-the art.

Concerning the technical evolution little has changed in the base technology used by the platform. The Java programming language and software development kit is still the most widely used technology for operating system independent server applications. Despite extensions and improvements of the Java EE standard, most web applications are still deployed on simple web servers like Apache Tomcat. Additional features provided by Java EE compliant and alternative application servers do not clearly seem to be worth the additional development and configuration complexity. Similarly, data persistence is still commonly implemented using relational database management systems accessed with O/R mapping tools like Hibernate instead of higher-level abstractions.

A big change, however, has occurred concerning the implementation of web applications. Five years ago, most web applications were, like the platform, based on frameworks like Struts, which provide a user experience more similar to web browsing than to native applications. Now, many frameworks have become available which support richer user interface components using extensive JavaScript libraries and asynchronous communication between client and server as supported in current web browsers. Products like Wicket [14] and Vaadin [15] replace the concept of web applications as a collection of interdependent web pages by an application-centered programming model similar to client application development. A new development of a platform with similar requirements would certainly profit from using such a framework both in terms of better user experience and in terms of more powerful and straightforward software development. Taking advantage of the increased capabilities of today's web browser clients (AJAX, HTML 5) would both improve user experience and decrease the operational demands on the platform server.

Nevertheless, the old-fashioned user interface technology is still sufficient for the current platform applications and therefore for consistency reasons nothing will be changed in the near future.

5.4 Lessons Learned

When looking back at the practical applications of the platform during the last five years, it becomes apparent that central parts of the platform have been useful in just about every single project so far. Other parts, however, have been only used once or mainly for demonstration purposes. Having these parts separated from the base modules of the platform would result in a leaner software product, which would be easier to extend and maintain.

In addition to a more restrictive selection of features to include in the platform, one may also question the effort made for platform-specific user interfaces. While the platform was initially intended to be usable by laypersons, in practice many parts of the platform are only used by the same small group of users, often the developers themselves. Instead of creating a large number of administrative user interface pages, it could have made sense to provide command line tools like SQL scripts and utility programs for administrative tasks. This would also have allowed automating tedious and error-prone tasks currently performed in the web user interface with a lot of point-and-click operations. Similarly, it would have been a good idea to stop extending the graphical interaction modeling system and concentrate on scripting language support as soon as it turned out that modeling was mainly left to the developers.

Concerning the implementation itself, the biggest change has been to stop using a full-fledged J2EE application server and to rely only on a web server. This simplified deployment and testing significantly. One might even have gone further and minimized the number of web applications maintained within the Eclipse development environment. In hindsight, the enforced modularity and installation flexibility gained by dividing the platform into several separated web projects does not seem worth the additional complexity and error proneness of having to deploy several interdependent web archives for a single installation.

6. CONCLUSION

The design and development of the Evolaris platform for mobile marketing has been a rewarding experience. Most decisions turned out to be right in the long run, especially the fundamental decision to rely on proven technology and established tools. The development process seems to have been adequate for the small number of its developers, although additional resources for quality assurance and stricter control over features to be built into the platform would have been advantageous.

Despite its comparatively long history, the platform now is still very much alive and its grown complexity does not yet prohibit improvements and extensions for new application domains. Also, even in the age of complex mobile phone client applications, server software coordinating and serving large numbers of mobile users is still very much needed. The platform can therefore be expected to be of good use for exciting new projects for years to come.

7. REFERENCES

[1] WURFL, Wireless Universal Resource File, http://wurfl.sourceforge.net.

[2] Groovy, An agile dynamic language for the Java Platform, http://groovy.codehaus.org.

[3] J2EE, Java EE reference page, http://www.oracle.com/technetwork/java/javaee/documentation/index.html.

[4] JBoss Application Server, http://www.jboss.org/jbossas.

[5] Apache Tomcat, an open source software implementation of the Java Servlet and JavaServer Pages technologies, http://tomcat.apache.org/.

[6] Hibernate, Relational Persistence for Java and .NET, http://www.hibernate.org/.

[7] MySQL, open source database, http://www.mysql.com/.

[8] Struts, open-source framework for creating Java web applications, http://struts.apache.org/.

[9] Eclipse, Java IDE, http://www.eclipse.org/.

[10] Apache Ant, Java library and command-line tool, http://ant.apache.org/.

[11] Gant, Groovy-based build system, http://gant.codehaus.org/.

[12] Jenkins, continuous integration, http://jenkins-ci.org/.

[13] Tellurium, automated testing framework, http://code.google.com/p/aost/.

[14] Apache Wicket, lightweight component-based web application framework, http://wicket.apache.org/.

[15] Vaadin, Java framework for building modern web applications, http://vaadin.com/home.