

027607

mGBL

mobile Game Based Learning

Specific Targeted Research Project

Information Society Technologies

D 5.5 –
Final integration report

Due date of deliverable: 30st September 2008
Actual submission date: 1st December 2008

Start date of project: 1. October 2005 Duration: 39 months

Evolaris
Mr. Thomas Ebner

Version 1.0

Project co-funded by the European Community within the Sixth Framework		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium	<input type="checkbox"/>
CO	Confidential, only for members of the consortium	<input type="checkbox"/>

1 Table of Contents

1	Table of Contents	2
1.1	General Deliverable Description	5
1.2	Revision history of this document	5
1.3	External Peer-Review	6
1.3.1	General description of the review process	6
1.3.2	Comments and Recommendations of the external reviewers	6
1.3.3	Changes	7
1.4	Research Background	7
1.5	Executive Summary of the Deliverable	7
1.6	Purpose	8
1.7	Target Audience	8
2	Platform Design	9
2.1	Requirements	9
2.1.1	Motivation for a Server-Side Platform	9
2.1.2	Stationary Clients	10
2.1.3	Technological Requirements	10
2.2	Technological Decisions	11
2.2.1	Base Technology	11
2.2.2	Toolkit	11
2.2.3	Server Technology	11
2.2.4	Middleware	11
2.2.5	Database	12
2.2.6	Web Application Framework	12
2.2.7	O/R Mapping Tool	12
2.2.8	Development Tools	12
2.2.9	Client Software	13
2.3	Platform Architecture	13
2.3.1	Overview	13
2.3.2	Web Interface Modules	14
2.3.3	User Management	15
2.3.4	Asynchronous Communication Modules	15
2.3.5	Database Modules	16
2.3.6	Reporting	16
2.3.7	Logging	17
2.3.8	Game Authoring	17
2.3.9	Game Management	17
2.3.10	Download Tool	18
2.3.11	Game Style Selection	18
3	Platform Installation	19
3.1	Server Installation	19
3.1.1	Java 2 Platform, Standard Edition, Runtime Environment	19
3.1.2	JBoss Application Server	19
3.1.3	MySQL Database	20
3.2	Deploying and Starting the Platform	21
3.2.1	Installing the mGBL Archive	21
3.2.2	Starting the JBoss Application Server	21

3.2.3	Accessing the Administrative Web Interface	22
4	Platform Development	23
4.1	Required Tools	23
4.1.1	Java Development Tools	23
4.1.2	Eclipse IDE and Plugins	23
4.1.3	Tomcat Web Server	24
4.1.4	JBoss Application Server	24
4.2	Checking out mGBL Eclipse Projects	25
4.2.1	Binary Projects	26
4.3	Building	26
4.3.1	Ant Scripts	26
4.3.2	Building within Eclipse	27
4.3.3	Deploying within Eclipse	27
4.3.4	Testing with the Tomcat Web Server	28
4.3.5	Testing with the JBoss Application Server	29
4.3.6	Creating a Standalone Archive	31
5	Development Guidelines and Concepts	32
5.1	Modules, Layers and Projects	32
5.2	Eclipse Technical Project Types	33
5.3	Code Style	33
5.3.1	Java	33
5.3.2	JavaScript	34
5.3.3	XML	34
5.4	Language	34
5.5	Code Comments	35
5.5.1	Java Source Code	35
5.6	The System Module	35
5.7	The Platform Project	36
5.8	Logging	36
5.9	Exception Concept	36
5.9.1	Principles	37
5.9.2	Base Exception Classes	38
5.10	Authorization Concept	39
5.10.1	Client Data Separation	39
5.10.2	Roles	39
5.11	Eclipse Project Folder Structure	40
5.12	Version Control	40
5.12.1	Subversion Folder Structure	40
5.12.2	Check-In Comments	41
5.13	Java Package Names	41
5.14	Other Folder Names	42
5.15	Building and Testing	42
5.15.1	Build Environment Project	42
5.16	Hibernate O/R Mapping	43
5.16.1	Hibernate Configuration	43
5.16.2	Hibernate Tool Application	46
5.17	Transaction Strategy	46
5.17.1	Transactions and User Interaction	47
5.17.2	Commit and Rollback	48
5.17.3	Locking	48

5.18	Referential Integrity	49
5.18.1	Automatic Creation of Restrictions.....	50
5.19	Data Layer Artifacts.....	50
5.19.1	Framework Database Project	50
5.19.2	Client-Projects with Data Layer Artifacts.....	51
5.20	Extending the Database	51
5.20.1	Restrictions.....	51
5.20.2	Rules.....	52

1.1 General Deliverable Description

WP number:	WP 5
WP name:	Technological Implementation
Deliverable number:	D 5.5
Deliverable name:	Final integration report
Responsible work package leader:	<p>name: Thomas Ebner address: Hugo Wolf Gasse 8-8a email: Thomas.ebner@evolaris.net phone: +43 (0)316 35 11 11 103 mobile: +43 (0)664 84 14 434 fax: +43 (0)316 35 11 11 200</p>
Involved project partners:	Evolaris, UM, RSA, Trieste

1.2 Revision history of this document

Date	Version	Description	Author
2008-11-18	0.1	Document created	Richard Hable
2008-11-26	0.9	Content complete	Richard Hable
2008-11-28	1.0	Peer review, improvements	Richard Hable

1.3 External Peer-Review

1.3.1 General description of the review process

The reviewer has been working in the area of Information Technology since 2005. His main Tasks include Software Requirement Specification and Development. After graduation from a vocational school (HTL) for electronic data processing and business organisation in 2004 he continued his studies at University of Derby and graduated in 2005 with a Bachelor Science (first class honours) in Computer Studies.

The review process included a critical view on the report.

1.3.2 Comments and Recommendations of the external reviewers

The report "D 5.5 – Final integration report" (Version 0.9) gives a good insight of the technical implementation and capabilities of the mGBL platform. It also describes the steps required to setup the platform for operational purposes as well a development environment to adapt or extend it. Development Guidelines and Concepts are also presented to retain the quality when being changed or extended.

In general the report is detailed, clear to understand and well structured.

Nevertheless, there are some points to mention:

- From the reviewers point of view the document misses a description which SMS/MMS Gateways are supported and how they can be configured.
- With having the web browser as main administrative client the reviewer proposes to include a list of supported browsers with minimum version numbers.
- For the reviewer in point 2.3.10 "Download Tool" it is not clear what software is meant and what it should be used for.
- In chapter 3 "Platform Installation" from the reviewer's point of view the requirement of Java and the recommended version should be mentioned.

- The reviewer also proposes to include a link to the referred “Sun Java Coding Guidelines” with a Version if provide in chapter 5.3.1.
- The reviewer proposes to recommend Check-In Comments (5.12.2) in English and not native language to improve the collaboration between developers.

1.3.3 Changes

The document has been changed and extended according to the recommendations of the reviewer.

1.4 Research Background

- D 4.1 Standards and technology monitoring report
 - All technology selection decisions have been considered
- D 4.2 Functional and technical specifications
 - All inputs for module structuring and required module functionality have been considered

1.5 Executive Summary of the Deliverable

This document describes the final version of the mGBL platform. All aspects of the created software artifacts are described concisely, allowing the technically savvy readers to get to know the motivation for the platform and to understand and use it and even to adapt the software according to their own needs.

This is also the base of the documentation given to users of the open source version of the platform. Detailed information about platform usage, however, is provided in separate documents together with the source code and binary distribution packages of the platform.

1.6 Purpose

This document provides information about the final version of the mGBL platform. This version contains all server-side software developed for the mGBL project, including the game selection tool and the game management and authoring tool. Binary templates of the games to be downloaded on mobile devices are also included.

Progress and results achieved during the three-year software development for the mGBL project are described. This includes information about design decisions; lessons learned during the course of the project; and, of course, the resulting software and its application.

Readers are provided with information which allows them to understand the software, know about the application possibilities, and to profit from the experiences made during software development for the mGBL platform.

1.7 Target Audience

This document is intended for everyone who wants to understand the technical implementation and capabilities of the mGBL platform. In practice, it provides administrators and software developers with information how to use the platform and possibly adapt it for their own needs.

2 Platform Design

2.1 Requirements

2.1.1 Motivation for a Server-Side Platform

At first sight, technological support for mobile learning mainly seems to require creating and distributing small programs to small mobile devices. However, looking at the use and proliferation of existing mobile software, from trivial games to complicated applications, we see that this conventional distribution and deployment method leads to a surprisingly low impact in practice. Use of certain standard built-in data services, especially SMS, is even more widespread than use of the Internet. Other software for mobile devices like personal digital assistants and mobile phones, however, is created mostly by small companies for a small number of customers. No common software in addition to built-in software provided by the manufacturers is established which can be built upon. Third-party applications work stand-alone with little impact on the overall improvement of mobile device user experience. Many people even completely ignore the option to add third-party software to their mobile devices. This is quite surprising; after all, today's modern mobile devices offer operating systems and hardware performance similar to expensive desktop systems available a few years ago. For example, the widely used mobile-phone operating systems *Symbian* and *Windows Mobile* support and require fast CPUs (hundreds of megahertz clock rate) and large main memory (dozens of megabytes).

It was therefore decided to create a platform intended to avoid this dead-end distribution of stand-alone applications in the domain of mobile game-based learning. Mobile game applications are considered to be only one type of artefact within an ecosystem of both server- and client-based software supporting mobile learning. Similarly, games are not considered solitary applications with fixed features, which can be used for a certain

amount of time, with more or less fun and profit, only to be thrown away and replaced by new ones. Instead, support for templates of games and game concepts is provided, which allows experts in the learning domain to create game flow and learning content without the need for additional technical support and implementation.

2.1.2 Stationary Clients

A large number of users, including administrators of the server platform, providers of learning content, and, to a certain extent, participants of games, have to be able to access the platform from their personal computers in addition to mobile devices. Therefore, it was important that users of personal computers should only need to install the minimum possible additional software. This goal could only be achieved by supporting all user interaction via standard internet browsers using advanced Web 2.0 technology such as *Ajax*. Special care had to be taken to create standard-compliant web pages usable on a large number of web browsers, among them the *Firefox* web browser of the open-source *Mozilla* project.

2.1.3 Technological Requirements

In order to efficiently develop server software supporting a large number of mobile games played concurrently, a decision for a middleware (i.e. software that connects software components or applications) standard was necessary. Microsoft Windows operating systems were almost completely dominant on personal computers at the beginning of the mGBL project and are still dominant today, despite the market-share growth of alternatives like Apple Macintosh and various Linux distributions. However, systems hosting server applications are usually based on other operating systems such as Linux or other UNIX derivatives. Therefore, an operating system independent middleware platform was required. The following aspects, in particular, were critical to the mGBL server software:

- web container infrastructure

- distributed communication
- transaction management
- relational database access.

2.2 Technological Decisions

In order to achieve as much operating system and vendor independence as possible, the platform is based mainly on open source tools and technologies.

2.2.1 Base Technology

Java technology was used wherever applicable in order to achieve

- device independence
- operating system independence
- industry-standard compliance
- modern object-oriented software development
- availability of powerful open-source tools
- widespread developer know-how
- extensive community support

2.2.2 Toolkit

The following Java 2 Platform, Standard Edition development kits were used in the beginning:

- Windows Platform - J2SE(TM) Development Kit 5.0
- Linux Platform - J2SE(TM) Development Kit 5.0

Several updates of the toolkit have been used in the meantime; we took care to remain compatible though.

2.2.3 Server Technology

Server technology according to the Java Platform, Enterprise Edition specification is used. This is the industry middleware standard for large, reliable server applications.

2.2.4 Middleware

The following web server is used as a servlet/JSP container:

- Apache Tomcat 5.5.15

Business components, which require advanced features like message-driven communication, are deployed on the following J2EE application server:

- JBoss Application Server Version 4.0.5

Both servers are widely-used open source software; professional commercial support is available.

2.2.5 Database

We use the widespread open source database MySQL as the main implementation and testing database.

- MySQL 5.0

2.2.6 Web Application Framework

The proven Struts framework is used for all web applications in order to achieve a unified user interface structure with clear model-view-controller (MVC) separation.

- Apache Struts 1.2.8

2.2.7 O/R Mapping Tool

The powerful object/relational mapping tool Hibernate is used for all database access. Hibernate supports automatic creation of Java data access objects and high-performance on-demand querying and data-access with caching. Thus, direct dependency of the business layer on the database system is avoided and object-oriented persistent data structures can be used.

- Hibernate 3.1.2

2.2.8 Development Tools

Development is based on the open-source framework Eclipse, which is an extensible integrated development environment. All platform modules are therefore organized within Eclipse project directories. The current major release of Eclipse, named "Ganymede", was used to create the final version of the platform.

2.2.9 Client Software

Because a large number of users has to be supported, users of personal computers should have to install as little additional software as possible. We achieved this by supporting all administrative access to the platform via a standard internet browser. Since most users have access to a Microsoft Windows personal computer with the Internet Explorer Browser installed, the mGBL web interfaces have been designed for this browser. In order to ensure standard compliance and take advantage of better debugging support, the web pages also were tested with the Firefox browser from the Mozilla project. The platform was continuously adapted when newer versions of these browsers became available. Currently, the following (minimum) browser versions are recommended:

- Mozilla Firefox 3.0.4 (Windows and Linux)
- Microsoft Internet Explorer 7.0 (Windows)

2.3 Platform Architecture

The mGBL project required a multi-user client/server software architecture with both mobile and stationary clients supported by an application server.

2.3.1 Overview

Figure 1 provides an overview of the system, containing the modules required for the project and their positions within an object-oriented, three-tier software architecture. Depending on its architectural position and functionality, a module can be implemented with one or more server applications, stand-alone client or mobile programs, or any software based on existing frameworks.

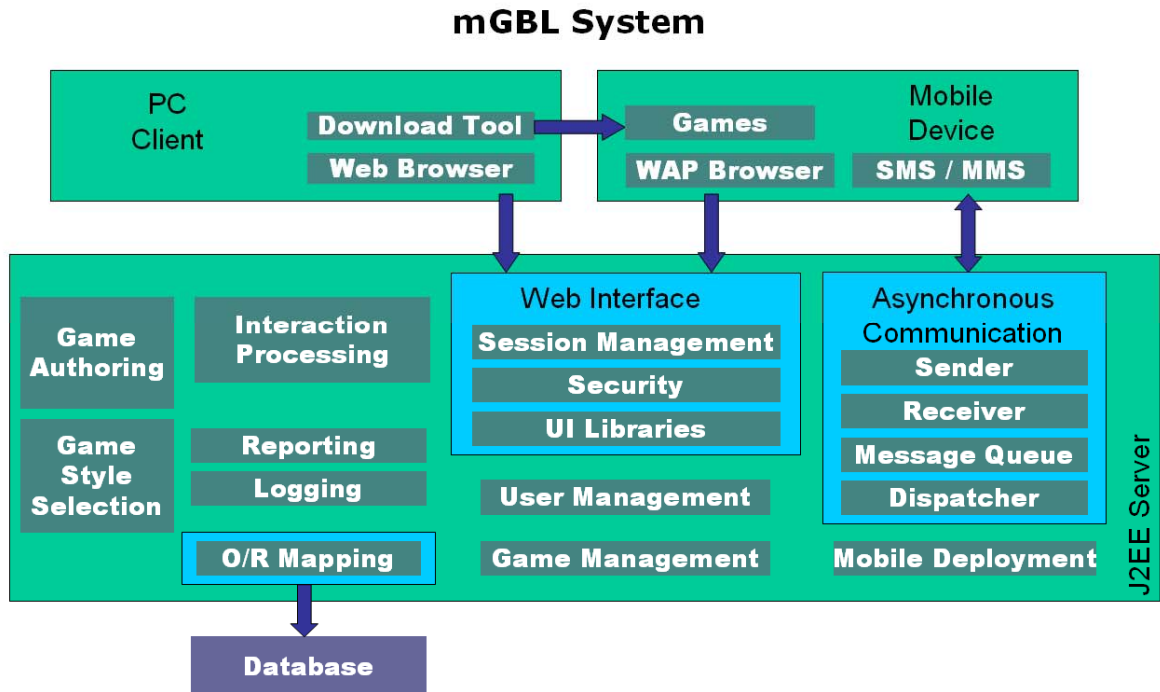


Figure 1: Platform Architecture

The modules described within the following chapters provide common services to allow for a consistent user experience and software structure.

2.3.2 Web Interface Modules

Users access the server mainly via a web browser. The following modules support providing consistent web pages for users with different roles:

- Session Management

User sessions have to be kept open during the exchange of stateless HTTP (or WAP) requests and responses.

- Security

Users may only access data and use functionality according to their rights and roles.

- User Interface Libraries, I18n

Common elements like style sheets and tag libraries are created to achieve a consistent user interface. A uniform mechanism for internationalization is provided to be used by all web pages within the project.

While not providing ready-made web pages, these modules contain support code and documents for all common web interface functionality within the mGBL system.

2.3.3 User Management

The system supports users with different roles and access rights. Therefore, it is necessary to administrate user accounts and provide support for other modules, which provide services for users. Since many users with similar roles exist, e.g. all pupils accessing a set of games, groups of users have to be supported for easier administration.

Thus, the following tasks are provided by the module:

- management of information about users, groups and their access rights to data and functionality
- support for authentication and authorization of users accessing mGBL web interfaces
- web user interfaces for administrative tasks like creating users and changing passwords

2.3.4 Asynchronous Communication Modules

In addition to the web interfaces provided for user interaction, the mGBL platform supports communication with mobile devices using asynchronous services like SMS and MMS. These messages may be sent directly to a user, e.g. to ask a question, which he or she is supposed to answer by sending an SMS. Mobile games may also automatically exchange asynchronous messages with the server, e.g. to record data about the learning process.

The following modules are provided:

- Sender

Sending of asynchronous messages is supported in such a way that the caller does not have to care about the technology used.

- Receiver

This module receives all incoming asynchronous messages.

- Message Queue

All incoming messages are collected in a queue for asynchronous handling.

- Dispatcher

Incoming messages have to be analyzed; invalid messages are ignored, other messages are forwarded to the responsible application.

2.3.5 Database Modules

Applications within the mGBL project are free to keep private data in any way they see fit, e.g. within XML files. Dynamic data on the server, however, always has to be stored in a central database in order to ensure consistent distribution of information. The following modules comprise the database support within the framework:

- Database

An industry standard relational database server is installed for high-performance transactional data access. Access to the database within the application server is performed using the SQL protocol.

- O/R Mapping

A mapping tool is used to provide an abstraction between object-oriented application and relational data layer. This allows the mGBL applications to access persistent data as a collection of structured interrelated objects.

2.3.6 Reporting

While the mGBL server is communicating with users and mobile clients, data is collected in the database. This data has to be visualized to provide

information like information about messages sent both manually and automatically.

2.3.7 Logging

Server applications within the mGBL system are required to create logs about their activities in a consistent way, which can be easily evaluated by the system administrator. The logging module provides support for different log levels, which allow categorizing log messages according to importance from purely informational messages up to critical errors, which require immediate attention.

2.3.8 Game Authoring

Usually, the learning content of a game is not static, but provided separately and regularly after the game template has been implemented. Both game templates and games with learning content have to be distributed to potentially many users and thus have to reside on the application server.

The game authoring module is a component, which allows authorized users to edit content for game templates and thus create games, which can be made available on the application server. Most authoring tasks are performed within the platform administrative web interface, which allows the user to quickly create content within their web browser without having to install software locally.

2.3.9 Game Management

The following functionality is available via a web interface:

- upload of game templates and games with content by authorized users
- definition of users and groups that may use them
- download of games by authorized users

2.3.10 *Download Tool*

Some mobile devices require game downloading from a directly connected personal computer. The software is device-dependent and usually provided by the vendor of the mobile device. Examples are:

- Sony Ericsson PC Suite for Smartphones
- Palm Desktop Software

2.3.11 *Game Style Selection*

The mGBL project supports the complete cycle of mobile game-based learning. This includes finding appropriate mobile game types for a range of learning purposes and for different learning modes. Therefore, a game style selection tool was created and included in the platform which allows specifying desired properties according to the following categories:

- learning content
- learning activities
- cognitive processes
- intelligences
- number of players.

After a game type has been selected, information about appropriate games is displayed including descriptions, screen shots and user comments.

3 Platform Installation

The platform is available on the SourceForge server in binary version. Thus, if the platform shall be installed without changes, it is sufficient to download and install the binary archives via the mGBL project page:

<https://sourceforge.net/projects/mgbl/>

Some tools have to be downloaded first, though, and some configuration is necessary to make use of the platform features. These tasks are described in the following chapters:

3.1 Server Installation

The platform and tools required by the platform are operating system independent and can therefore be installed on a wide variety of Unix-like and Windows operating systems. Linux and Windows XP have been used during development of the platform.

3.1.1 Java 2 Platform, Standard Edition, Runtime Environment

The following (minimum) version of the Java Runtime Environment should be installed:

Java SE Runtime Environment (JRE) 6 Update 10

It can be downloaded for different operating systems via the following URL:

<http://java.sun.com/javase/downloads/index.jsp>

3.1.2 JBoss Application Server

The JBoss application server software, version 4.0.5, can be downloaded via the following URL:

<http://www.jboss.org/jbossas/downloads/>

After extracting the archive on the local development machine, some changes have to be performed on the extracted files. In the download section of the SourceForge project page (select package *mgbl*), the archive file *platform-configuration.zip* has to be downloaded and extracted within the JBoss directory. This will add and modify some files.

3.1.3 MySQL Database

The platform is preconfigured to use a MySQL database instance to store all data. It is recommended to use the latest 5.0 version of the MySQL server and an appropriate version of the administration tool "MySQL Administrator" including the MySQL command line client. MySQL software can be downloaded via the following URL:

<http://dev.mysql.com/downloads/mysql/5.0.html#downloads>

The free version of the MySQL server can be found for different operating systems under the name "MySQL Community Server", the administration tool within "GUI Tools". When installing the database make sure to use "UTF-8" as default character set and "InnoDB" as default database engine.

A database schema and a database user with appropriate access rights have to be created with the MySQL Administrator tool:

- Select item "Catalogs"
 - Create a new database schema "framework" with context menu item "Create New Schema" in the schema view.
- Select item "Users"
 - Create a new user "framework" with context menu item "Create New User" in the users view
 - Use password "evolaris" if you want to use the predefined settings of the platform; otherwise select your own password and set environment variable "hibernate.connection.password" when starting the application server.
 - Assign all privileges to access the framework database within tab "Schema Privileges"
 - Create a new host "localhost" with context menu item "Add host from which the user can connect" on the framework user.
 - Assign all privileges to access the framework database within tab "Schema Privileges"

Some initial tables and views have to be created for the mGBL platform to work. In the download section of the SourceForge project page (select package *mgb*), the archive file *mysql-configuration.sql* can be downloaded and executed e.g. within the MySQL Administrator tool:

- Open the MySQL command line client with menu item "MySQL Command Line Client" of menu "Tools".
 - Select the framework database with: *use framework*
 - Paste the entire content of the *mysql-configuration.sql* via the clipboard into the console.

Note that this script creates a single platform user *administrator_mgb* with password *administrator*, who is assigned all roles and thus can be used to create additional users and groups with arbitrary roles. The password should be changed on production servers to something less easy to guess.

If you need a clean database later, remove the framework schema and recreate it in the administrator tool as described above and execute the configuration script again.

3.2 Deploying and Starting the Platform

3.2.1 Installing the mGBL Archive

The archive file *MgbLEAR.ear* can be downloaded from the SourceForge server. Note that this is a zipped archive; thus, it can be examined and even changed using any standard tool for ZIP archive management.

The file has to be copied into the *server/default/deploy* directory of the JBoss installation. All platform services will then be available once the application server is started.

3.2.2 Starting the JBoss Application Server

The application server can be started via scripts in its *bin* directory. On windows systems the *run.bat* batch file has to be executed, on other systems the *run.sh* shell script has to be invoked.

3.2.3 Accessing the Administrative Web Interface

The following URL leads to the start (login) page of a locally installed platform:

<http://localhost:8080/platform/secure/index.do>

4 Platform Development

The main parts of the mGBL platform are available as open source. This allows interested developers to modify and extend the platform according to their own needs. The following subchapters describe how to set up and use the development environment.

4.1 Required Tools

In addition to a full installation of the runtime environment (see chapter 3) several development tools are required to make changes and create modified versions of the platform.

4.1.1 Java Development Tools

The latest version of the Java development tools should be downloaded from the Sun website:

<http://java.sun.com/javase/downloads/index.jsp>

Note that the platform was developed with compiler compliance level 1.5. Thus, earlier Java 5 versions would probably work too.

4.1.2 Eclipse IDE and Plugins

Eclipse has to be used as integrated development environment. The Ganymede bundle for JEE development has to be downloaded, service release 1 (SR1) or higher:

<http://www.eclipse.org/downloads/>

In addition to the plugins delivered within this release, a subversion plugin is required to access the source repository. This can be installed with the software update dialog accessible within the Eclipse help menu. The following site has to be added:

http://subclipse.tigris.org/update_1.4.x

All available plugins should be selected in the install dialog.

In order to work with platform projects, the following Eclipse settings are required within menu "Window", menu item "Preferences":

- General / Workspace: Text file encoding "UTF-8"
- Java / Compiler: Compiler compliance level: 1.5
- Server / Runtime Environments: add JBoss v4.0 with server directory according to the local JBoss 4.0.5 installation.
- Server / Runtime Environments: optionally add Apache Tomcat 5.5 with server directory according to the local Tomcat installation.
- XDoclet: set "Version" to "1.2.3" and "XDoclet Home" to directory "MgblBuildEnvironment/lib" within the workspace (path can be only selected after the mGBL Eclipse Projects have been checked out as described later).
- XDoclet / ejbdoclet: select JBoss
- Web / JSP Files / Validation: switch off "Validate JSP fragments" in order to avoid unjustified error messages.
- Validation: switch off "JSP Content Validator" and "JSP Syntax Validator" for both "Manual" and "Build" in order to avoid unjustified error messages.

4.1.3 Tomcat Web Server

While not absolutely necessary, it is recommended to install the Apache Tomcat Web Server on the local development machine. This allows much faster turnaround cycles during web development than the JBoss application server. The latest 5.5 version of Tomcat should be downloaded from the following site:

<http://tomcat.apache.org/download-55.cgi>

The downloaded archive can then be extracted to an arbitrary position on the local development machine.

4.1.4 JBoss Application Server

Version 4.0.5 of the JBoss application server has to be installed for local testing. If it has already been installed as described in chapter 3, the

following steps are only required if differently configured versions of the platform shall be deployed.

The JBoss application server can be downloaded via the following URL:

<http://www.jboss.org/jbossas/downloads/>

After extracting the archive on the local development machine, the following additions and changes have to be performed in the JBoss directory:

- Copy the following jar-files from the *lib* directory of project *MgblBuildEnvironment* into the *lib/endorsed* directory: *asm.jar*, *c3p0-0.9.1.2.jar*, *cglib-2.1.3.jar*, *mysql-connector-java-3.1.12-bin.jar*
- Copy *framework-mysql-ds.xml* from the *setup* directory of project *MgblDatabase* into the */server/default/deploy* directory.
- Uncomment the single-sign-on line in */server/default/deploy/jbossweb-tomcat55.sar/server.xml* by removing the first and third of the following lines:

```
<!--  
    <Valve className="org.apache.catalina.authenticator.SingleSignOn" />  
-->
```
- Copy *platform-login-config.xml* from the *EarContent/META-INF/* directory of project *MgblEAR* into the */server/default/conf/* directory.
- Copy *platform-login-config-service.xml* from the *EarContent/META-INF/* directory of project *MgblEAR* into the */server/default/deploy* directory.
- Copy *scheduler-service.xml* from the *setup* directory of project *MgblSchedulerEnqueuer* into the *server/default/deploy* directory.

4.2 Checking out mGBL Eclipse Projects

All Eclipse projects required for building the platform are available via the SourceForge Subversion repository. Checking out is possible within Eclipse with menu File, item "Import...", import source "SVN / Checkout

Projects from SVN". The following repository location has to be created or selected:

<https://mgb1.svn.sourceforge.net/svnroot/mgb1>

All folders within "platform/trunk" are platform Eclipse projects and can be checked out into the Eclipse workspace by selecting and pressing button "Finish".

4.2.1 Binary Projects

The following projects are not available in source form but are required to build a working version of the platform:

- Mgb1Async
- Mgb1AsyncDispatcherWeb
- Mgb1SMSServices
- Mgb1SMSServicesWeb

Therefore, they have been checked into the SVN repository without Java source files. Instead, correctly built class files are checked in within the build directory. In order to preclude the Eclipse Java compiler from deleting class files whose sources are not available, the Java builder has been renamed within the project configurations.

4.3 Building

4.3.1 Ant Scripts

Although all Java sources are compiled within the Eclipse IDE, some preparational work is performed via manually written Ant scripts. In order to easily invoke Ant tasks, add the Ant view to the Eclipse Java perspective with menu "Window", menu item "Show View.../Ant". Copy the following Ant build scripts into this view with drag and drop:

- Mgb1Database / setup / build.xml
- Mgb1EAR / setup / build.xml

Use target "rebuild_db" from the Mgb1Database build script to setup a local test database with fixed database schema and required initial configuration data. Target "insert_data" from the Mgb1EAR build script is

used to create additional test data, e.g. test users. Note that these entries are provided for demonstration purposes only; in order to test the platform locally additional user accounts have to be created either directly within the database or via extensions to the Ant build scripts.

The Ant target "pojos" from the MgbDatabase build script has to be invoked in order to create or recreate database access objects from Hibernate mapping files. This is necessary after checking out the database project and every time the mapping files are modified. The directory "gen" within the MgbDatabase project has to be refreshed, either manually or automatically via the Eclipse preference settings, after invoking this target. Note that the files in this directory must not be checked into the SVN repository!

4.3.2 Building within Eclipse

The complete workspace can be built with menu "Project", menu item "Clean...", by selecting "Clean all projects". A complete build will be performed immediately. If the recommended preferences are set, no errors (but plenty of warnings) should be reported in the "Problems" view.

4.3.3 Deploying within Eclipse

While it is possible to completely test and deploy the built mGBL application within a separately started JBoss application server, shorter turnaround cycles and better debugging support are possible if the web or application server is started within Eclipse.

This can be done within the Eclipse server view, which can be opened with menu "Window", menu item "Show View / Other" and selection of "Server / Servers" within the resulting dialog.

Within the server view a context menu item "New / Server" is available which allows adding the configured web or application servers to the view. The Eclipse development environment automatically creates a server project "Servers" within the workspace containing the server configuration if no server project exists in the workspace. The mGBL

distribution, however, already contains a server project with the name "MgbIServers", which is automatically used for added servers.

Once the JBoss application server and possibly also the Tomcat web server have been added to the server view, mGBL projects can be added to a server with context menu item "Add and Remove Projects...". Note that the mGBL server project already contains a correct "server.xml" file for testing mGBL within the Tomcat web server. Every time projects are added to or removed from this server, automatic changes are performed on the file which have to be undone with context menu item "Team / Revert...".

4.3.4 Testing with the Tomcat Web Server

Only the administrative web interfaces can be tested within the web server. Other features like timer events require message queues and EJBs which are only supported by JEE application servers like JBoss.

The following web projects have to be added in the server view:

- MgbIPlatformWeb
- MgbIWeb

The mGBL project additionally requires many JAR files which are not included in these web projects. These have to be added to the classpath of the launch configuration of the server. The launch configuration can be opened with menu "Run", menu item "Run Configurations...", selecting "Apache Tomcat / Tomcat v5.5 Server at localhost". There, the classpath tab and then the "User Entries" node have to be selected. With the "Add JARs..." button a dialog is opened, where all files below "MgbIBuildEnvironment / tomcatlibs" have to be selected and added to the classpath. Additionally, the following Eclipse projects have to be added with the "Add Projects..." button:

- *MgbISystem*
- *MgbIDatabase*
- *MgbIUserManagement*
- *MgbIAsync*

- *MgblAsyncDispatcherWeb*
- *MgblBlogging*
- *MgblBloggingWeb*
- *MgblGameManagement*
- *MgblGameSelection*
- *MgblPlatformWeb*
- *MgblSMSServices*
- *MgblSMSServicesWeb*
- *MgblSystemWeb*
- *MgblWeb*

Now the web server can be started by selecting it in the server view and pressing the start button. Once the server is running, the following URL can be used to access the administrative web interface:

<http://localhost:8080/platform/secure/index.do>

4.3.5 Testing with the JBoss Application Server

The complete platform functionality can be tested locally. Note, however, that correct database configuration and a contract with one of the supported general-purpose SMS gateway providers, SMS.AT and VeriSign, are required to use SMS sending and receiving. The following attributes have to be configured in table *T_SMSSVC_SENDERS*:

SMS.AT

- GATEWAY_PROVIDER: *SMSAT*
- MSISDN: the mobile phone number received from SMS.AT (e.g. 431234567)
- SMSAT_USERNAME: the username received from SMS.AT
- SMSAT_PASSWORD: the password received from SMS.AT

VeriSign

- GATEWAY_PROVIDER: *3UNITED*
- MSISDN: the mobile phone number received from VeriSign (e.g. *431234567*)
- THREE_UNITED_APPLICATION_ID: the application ID received from VeriSign
- THREE_UNITED_AUTH_KEY: the authorisation key received from VeriSign

Also, in order to allow sending and receiving e-mail messages, a valid mail server URL has to be configured in table T_SYS_CONFIGURATION.

The enterprise archive *MgbIEAR* has to be added to the JBoss server configuration in the server view with context menu item "Add and Remove Projects...". Then the JBoss application server can be started with the start button. Eclipse will automatically create an EAR file within the configured JBoss server directory. Depending on the performance of the development machine, the default timeout for starting the application server may be too low and lead to an error message dialog. In this case, the timeout should be increased to several minutes within the JBoss configuration page, which can be opened by doubleclicking on the JBoss entry in the server view. Enter e.g. value *999* as start time and value *200* as stop time within the "Timeouts" section and then save the page.

After JBoss has been started successfully, the platform administrative interface can be accessed via the following URL:

<http://localhost:8080/platform/secure/index.do>

The platform will also perform timer events and react to incoming events. If timer events shall be used, a JAR file "SchedulerEnqueuer.jar" has to be created from the *SchedulerEnqueuer*, *AsyncEJBClient* and *SMSServicesEJBClient* projects and copied into the *server/default/lib* directory of the JBoss installation. This is only necessary once and can be done within Eclipse while JBoss is not running:

- Invoke menu item "Export..." of the context menu of the *MgbSchedulerEnqueuer* project and select "Java / JAR file".
- In addition to the preselected *MgbSchedulerEnqueuer* resource select *MgbAsyncEJBClient* and *MgbSMSServicesEJBClient*.
- Choose as destination file path the path name *server/default/lib/SchedulerEnqueuer.jar* within the JBoss installation directory.
- Press button "Finish" and ignore warnings about duplicate entries.

4.3.6 Creating a Standalone Archive

In order to create a deployable EAR file, menu item "Export..." has to be selected in the context menu of project "MgbIEAR". There, export destination "Java EE / EAR file" has to be selected and "MgbIEAR.ear" should be chosen as destination file within the JBoss "server/default/deploy" directory.

Starting the JBoss application server will then allow using the new mGBL platform version independently from Eclipse.

5 Development Guidelines and Concepts

In order to retain a manageable architecture and allow collaboration between independent developers, some rules are necessary when modifying or extending the mGBL platform. The following subchapters provide guidelines for developers who do not only want to understand the platform but also want to make changes or extend the platform without impairing its quality.

5.1 Modules, Layers and Projects

The main structuring of our software is the division into *layers* (tiers). For every part of software, it is clear which layer they belong to. This determines the technology used for implementation and the rules to be obeyed during development. Layers are clearly distinguished by package and folder names – see the description of Java package names within the following chapters.

In addition to the layers, the software is divided into *modules* according to the *client project* requirements. Modules may contain both project-specific and general (framework) parts. A special client project *platform* is used for all functionality which is deployed only once but provides functionality for different client projects.

Within the development environment, there is an additional division into *Eclipse projects*, which differ according to tools (e.g. editors, views and builders) and configuration needs. Independently of the Eclipse project division, the Java package structure also creates unique distinctions between classes belonging to different client projects, modules and layers.

A module can be implemented within one or more Eclipse projects.

5.2 Eclipse Technical Project Types

In order to take advantage of the available Eclipse plug-in modules (mainly Web and J2EE standard tools) the following project types for source code are used:

- Java and J2EE Utility Project (no special suffix, e.g. *UserManagement*)
- Dynamic Web Project (suffix *Web*, e.g.: *PlatformWeb*)
- EJB Project (suffix *EJB*, e.g.: *AsyncEJB*)
- EJB Client Project (suffix *EJBClient*, e.g. *SMSServicesEJBClient*)

In addition to these project types, projects of the following types are required for configuration and deployment:

- Server Project (Suffix *Servers*, e.g.: *MgbIServers*)
This project is created with the “Move to Workspace” context menu entry within the Eclipse Servers view. It has to be added to the version control system in order to keep track of server configurations.
- Enterprise Application Projects (suffix *EAR*, e.g. *MgbIEAR*)
This project contains references to projects which are packaged into a single EAR file.

5.3 Code Style

It is important that all developers use the same code style and file format settings in order to achieve generally understandable and interchangeable source code.

5.3.1 Java

We adhere to the Sun Java Code Conventions with the following exceptions:

- Indentation is always performed by tabs.¹
- The maximum line length must be at least 200 characters.²

¹ It is up to the developers to select the number of spaces displayed for each tab. Adjust spaces/tab settings within the General/Editors/Text Editors tab in the Eclipse preference dialog.

The code conventions are available via the following URL:

<http://java.sun.com/docs/codeconv/index.html>

5.3.2 JavaScript

The same code style as for Java also applies to JavaScript where possible. Thus, also JavaScript code has to be written with tab indentation and one statement per line.

5.3.3 XML

XML files use the same indentation and line-length rules as Java and JavaScript; however, it is allowed to use several elements within one line. If the begin tag of an element is provided in a separate line then also the end tag must be in a separate line.

For example, the following structure is allowed:

```
<element1 attribute1="xyz">
    <element2 attribte2="xyz"/>
    <element3 attribte3="xyz"><element4/></element3>
</element1>
```

The following structure is not allowed:

```
<element1 attribute1="xyz"><element2 attribte2="xyz">
</element2></element1>
```

5.4 Language

We use English for all identifiers, comments and documentation unless explicitly noted.³

Source files usually only contain characters from the ASCII character set. Only sources which are not internationalized and property files for other

² This way we avoid splitting lines too often. Adjust this setting by modifying the used profile within the Java/Code Style/Formatter tab of the Eclipse preferences – tab “Line Wrapping”.

³ This is necessary because we cooperate with international project partners.

languages may require additional characters. In these cases UTF-8 encoding should be used where possible.⁴

5.5 Code Comments

5.5.1 Java Source Code

Java source files may only be checked in when they are commented according to the following rules:

- A valid *JavaDoc* comment has to be provided for each *public class* describing the purpose and features of the class.
- Valid *JavaDoc* comments are required for all *public* and *protected* member *variables* and *methods*.
- At least a single-line comment has to be provided for each *private* method.
- For each *method parameter* the behavior with special values like null or empty lists/sets has to be documented, if allowed.⁵
- For each *return value* which may be a special value like null or an empty list/set, the meaning of this value has to be documented.⁶
- As an exception to these rules, simple Java bean *getters* and *setters* do not have to be commented.

5.6 The System Module

A special module *System* is provided for general functionality. It is implemented within the Eclipse projects *System* (business layer) and *SystemWeb* (web layer). The *System* module contains support for common framework functionality like menu bars within web pages.

⁴ Unfortunately, the Struts framework does not work well with UTF-8 encoded property files. Therefore, we currently use ISO-8859-1 encoding for property files.

⁵ If no description of special values is provided, it is assumed that the caller must not pass special values.

⁶ If no description of special values is provided, it is assumed that the method never returns special values.

5.7 The Platform Project

Among the client projects, a special project *platform* is used for general services which are provided for several clients.⁷

Also, other client projects may refer to platform web pages via their configurable menus.

5.8 Logging

In order to be able to control platform activities, server software has to log using the Java *Log4j* library. The following rules apply:

- All changes to the database have to be logged.
- The usual categorization into DEBUG, INFO and WARN messages is used.
- Variable parts of logging messages have to be separated from static parts by apostrophes (accent grave) if there is danger of confusion.⁸

5.9 Exception Concept

Error handling is a central issue spanning all components and layers of a multi-tier enterprise application. We use an exception concept applicable both for user interaction and autonomous processes in the background.⁹

⁷ One example for this is the possibility to send an SMS message to several users at the same time.

⁸ E.g. User `John Smith` deleted the database.

⁹ The following requirements had to be fulfilled:

- The normal flow of operation shall be disturbed as little as possible by exceptional cases.
- Adequate and comprehensive information about errors shall be made available to end users, system administrators and programmers.
- Messages for users have to be localized, i.e. translated into any of the languages supported by the respective application; system administrators and programmers are supposed to know English.
- It must be clearly deductible for every exception, which recovery options (e.g. retry, give up) are available for both application and the end user.

5.9.1 Principles

We adhere to the following design decisions:¹⁰

- A proper Evolaris exception subhierarchy has been introduced, providing classes which support storing messages and exception type information according to the above requirements.
- Checked exceptions from external classes are caught as early as possible and wrapped into Evolaris exceptions, if not immediately recoverable.
- All Evolaris exceptions are unchecked, i.e. they are directly or indirectly derived from class *java.lang.RuntimeException*, in order to not require verbose *throws* clauses within the entire method caller hierarchy.¹¹
- Evolaris exceptions are caught and reacted upon as late as possible; i.e. within the top level of web interface or EJB code. They must *not* be used to signalize situations to be reacted upon within some intermediate method call level.
- No wrapping of Evolaris exceptions is done – all information required for handling the exception therefore has to be collected and stored in the exception object at the position of recognition of the exceptional case.
- All Evolaris exceptions are based upon a base class *GeneralException* containing separate string fields for user, administrator and programmer messages. The catcher of an exception evaluates these fields and decides about the appropriate way of distributing the information, e.g. by writing into log files or displaying an alert.

¹⁰ These decisions are based on the technical possibilities of the Java programming language and the J2EE application environment.

¹¹ A switch from checked to unchecked exceptions has also been made by popular Java frameworks such as Swing and Hibernate 3. Evidently, the more precise method interface definition with checked exceptions did not prove to be worth the subsequent interface complexity and instability.

- The type hierarchy of Evolaris exceptions is organized according to the recovery options available for the catcher.

5.9.2 Base Exception Classes

Currently, the following exception classes are defined.

- *InputException*
This exception is used to indicate errors based on invalid user input. Usually, it contains a localized user message providing the user with information on how to change their input to avoid the problem.
- *ConfigurationException*
This exception is thrown when an external system is not accessible and/or there is some problem with settings in configuration files or the database. Since the user can do nothing about that, they should be provided with a general error message and the hint to try again later.
- *BugException*
This exception is thrown if a programming error has been detected. This is usually the case when a method is called with invalid parameters or some other unexpected situation has occurred. A general error message should be presented to the user.
- *InteractionException*
This exception is thrown when logical modelling errors are recognized during interaction list processing. Both a localized user message and an informative message for the administrator shall be provided. The user message is shown when the interaction list is invoked within the administrative platform interface; the administrator message is needed when services are performed in the background.

These exceptions may be used directly or may be extended by module-specific exception classes, if additional exception information or more differentiated exception types are required.

In any case, exceptions have to be logged in the web or application server log files with all information available (including stack trace of the call hierarchy) in order to allow for the programmer or system administrator to find out about the reason of the exception and possible corrections.

5.10 Authorization Concept

The authorization concept is used to give different users different rights on functions and data.

5.10.1 *Client Data Separation*

Users are always assigned to exactly one group and normally also may only access data within their group. The only exception to this rule applies to user with role administrator (see below).

5.10.2 *Roles*

A user can have one or more *roles*. Depending on his roles the user is allowed to access functions and data or not. If a user is assigned to more than one role he has the sum of all rights of both rules.¹²

The following roles are important for all projects. See the descriptions in the platform administrative user interface for information about other roles.

user

This role is used to give the user the possibility for a web login. Users that are not assigned to this role are not allowed to use the platform web application.

administrator

Users within this role are allowed to do group spanning operations. This means this role gives you access to all groups and their data.¹³

¹² E.g. if the user is in role *sms_sender* and also in the role *report_viewer* the user is allowed to send SMS messages to user sets of his own group and to view all reports of his own group. If the user is also in role *administrator* he is allowed to send messages to any user set in the system and he is also allowed to view all reports of any group in the system.

group_administrator

This role is used for group administration. This means users within this role are allowed to access all user data within his group.

5.11 Eclipse Project Folder Structure

The folder structure within eclipse projects is chosen based upon the following principles:

- congruence with the default Eclipse project structure
- clear separation between source files (*src*), test source files (*tst*), generated files (*gen*) and runtime files (e.g. *build*)¹⁴
- unique package structure for all Java code independently of eclipse project segmentation

Additionally, some generally required files like configuration files maintained by Eclipse are stored directly within the project directory.¹⁵

5.12 Version Control

All files needed to create deployable software, including configuration files and required third-party libraries have to be stored in the version control system.

5.12.1 Subversion Folder Structure

The standard Subversion folder structure is used (see Subversion documentation for more details). Thus, all up-to-date sources are stored within the *trunk* folder. Tagged versions and branches are stored in the *tags* and *branches* folders.

¹³ E.g. if the user is allowed to send SMS messages and he has also the role administrator he is allowed to send messages to any user set in the system. The user is not restricted to his own group. Otherwise access to data is always limited to the own group of the user. So this role is not only used to give access to functionality but it is also used to grant access to additional data.

¹⁴ Within the project directory there are separate folders for source and generated code.

¹⁵ Note that not all directories and files are visible within the Eclipse Package Explorer view. Even within the Navigator view, files and directories whose names start with a dot are usually hidden by the default filter configuration.

5.12.2 Check-In Comments

The following rules apply when checking in one or more files into the repository:

- Comments are optional and may consist of arbitrary text describing the changes performed.
- Comments should be written in English.

5.13 Java Package Names

Package names are used according to the Java development guidelines, i.e. they consist entirely of lower-case characters. We do not use the old-fashioned *com* or *org* prefixes.

- The first component of the package name is always the name of the implementing organization – i.e. *evolaris* for software created by Evolaris.
- The second component of the package name indicates the project name. For code usable in more than one project, the name *framework* is used.
- The module name is used as third component of the package name. E.g.: *async* for the Async module.¹⁶
- The fourth component indicates the layer the code belongs to. Each layer belongs to a single tier within an enterprise application. Currently the following layer names are used:

- *web* – code used to implement a web interface (client tier)
Code within these packages should communicate only with code from the business layer.
- *business* – code performing actual program logic (middle tier)
This code may communicate with the data layer and external systems.¹⁷

¹⁶ Currently, most module package names are abbreviations of module names; e.g. *um* instead of *usermanagement*. We keep using these abbreviations for existing modules; however, the names of new modules may never be abbreviated.

- *datamodel* – the interface to the database (middle tier)
This is the package for Hibernate configuration files and classes generated by Hibernate tools. We collect all these files in a single *MgblDatabase* project.
- *communication* – the interface to external systems (middle tier)
This layer contains EJBs and classes which support communication with external systems, e.g. by sending emails.
- *util* – utility classes (no specific layer)
This is the package for general classes like transaction and exception wrappers.

Further package components may be used to structure classes according to the needs and complexity of the tasks to be performed.

5.14 Other Folder Names

Resources other than Java classes should be stored in a folder structure similar to Java packages. It is, however, often not possible to clearly assign resources like web pages to a single module. Therefore, instead of the module name, folder names like *sys* and *layout* can be used.

5.15 Building and Testing

We use the Eclipse Java development environment and available plug-ins wherever possible to achieve an efficient development process with fast turn-around cycles.

5.15.1 Build Environment Project

Many external libraries from open-source frameworks and tools are required when building projects and when running applications. In order to achieve consistent library version usage most of these libraries are

¹⁷ Note that we do not use a separate layer for data access objects. Thus, business layer methods may be as simple as loading and storing data objects using Hibernate library methods.

collected as JAR files within a single Eclipse project *MgblBuildEnvironment* which is referred to by most other projects. Since the sole purpose of this project is to hold external libraries, it does not contain any source code or created classes. All libraries are contained in a single *lib* folder.

Since library files are required for building working applications, they have to be stored in the version control system just like source files.

5.16 Hibernate O/R Mapping

A single database is used for all client projects within a platform instance. It is even possible for more than one platform instance to share a single database.

We use the O/R mapping tool Hibernate for database access within Java.¹⁸

5.16.1 *Hibernate Configuration*

Hibernate has to know how to access the database and how to translate database tables into Java classes.

The main configuration file is *hibernate_framework.cfg.xml*, which contains JDBC connection parameters and references to Hibernate mapping files. We use one mapping file per Java data object class or class hierarchy to be created by the Hibernate tools. Usually there is also one table per data object class. However, Hibernate creates additional tables for *many-to-many* mappings and class hierarchies.

Mapping Rules

¹⁸ This allows to access different repository types (databases) via uniform Java classes and high-level methods with very little configuration changes. Hibernate tools also automatically create Java data objects. Thus, no additional effort is required to encapsulate e.g. SQL statements or to create data access objects.

The following rules have to be obeyed when defining the mappings between database tables and Java data objects.

- Table names obey a strict naming convention using underscore characters for word division (similar to constants in the Java naming convention):

T_<MODULE AND/OR CLIENT PROJECT>_<NAME> (e.g. *T_ASYNC_MAILBOXES*)

Note, that the plural is used for tables intended to contain more than one entry (i.e. most tables). In order to avoid problems with restrictions of the Oracle database, table and attribute names must not be longer than 30 characters. Thus, abbreviations of table name parts may be necessary, e.g. table *T_SMSSVC_EMAIL_CES* contains entries of type *EmailCommandEntry*.

- Data access classes based on tables are named like the table names without the prefixes according to the Java naming convention for classes, e.g. *Mailbox*. The module information is already contained in the package name (see package name guidelines). Note that data access class names are always in singular form.
- Each class contains a field *id* of type *long* which corresponds to a primary key attribute *ID* in the database table. ID values are automatically assigned with the Hibernate sequence generator.
- Attribute names are assigned according to the Java naming conventions for member variables, the corresponding database attributes according to the Java naming convention for constants, e.g. *firstName* -> *FIRST_NAME*.

It is not allowed to use attribute names which are keywords in either the MySQL or the Oracle database. Thus, e.g. instead of the attribute name *comment*, the name *description* has to be used.

- Strings must always be specified with a defined maximum length, because default lengths may differ for different database systems.

- Restrictions like *not-null* and *unique* are always assigned for both class properties and database attributes.
- Attributes referring to other classes are implemented with foreign-key database attributes containing the ID of the referred-to table. The following naming convention is used: *FK_<DESTINATION TABLE NAME>_ID* (e.g. *FK_GROUP_ID*) If additional information is required to understand the meaning of a foreign-key attribute or if more than one foreign key to the same target is required, an additional name component has to be inserted. (e.g. *FK_SENDER_GROUP_ID, FK_RECEIVER_GROUP_ID*)
- Foreign key restrictions and indices are automatically created by Hibernate. Some existing restrictions and indices are named according to a naming scheme similar to foreign key attributes. Now we use the names which are automatically assigned by Hibernate. If one of these names is shown in an error message, its meaning can be found out by checking the SQL statements used for table creation.
- Association tables (*many-to-many*) are explicitly assigned a table name according to the following convention:
T_<MODULE_NAME>_<SOURCE_NAME>2<DESTINATION_NAME>
(e.g. *T_MGBL_GAME2LABEL*)
- The not-null restriction has to be used whenever reasonable; both within the *property* and the *column* element.
- Not-null properties have to be represented by primitive Java types whenever possible.¹⁹ E.g.: use *type="int"* instead of *type="integer"* or *type="java.lang.Integer"*.
- Null properties have to be represented by Java class types.²⁰ E.g.: use *type="java.lang.Integer"* instead of *type="integer"* or *type="int"*.

¹⁹ This way we avoid uncertainty about null checks within Java code.

²⁰ This allows checking for null values in the database.

- Boolean properties have to be represented as integer values (0 = false, 1 = true)²¹ in order to avoid problems with different database-specific representations. Usually, these properties are *not-null*, thus, `type="int"` should be used.
- Enumeration values have to be represented as integer values similar to boolean properties.²² Usually, these properties are *not-null*, thus, `type="int"` should be used.

5.16.2 *Hibernate Tool Application*

Hibernate tools are available for usage within ANT build scripts and as Eclipse plug-in (Hibernate console)²³. See the chapter about script-based building and setup for more details.

5.17 Transaction Strategy

We use a unified transaction strategy which is a compromise between performance and security and should be sufficient for most applications. It is based on the following principles:

- Short transactions
- Pessimistic locking²⁴

Access to the database via Hibernate always requires an active transaction—even if only read operations are performed. Therefore, database operations are always surrounded by start and commit transaction calls. In case of an error, it is always necessary to invoke a rollback of the current transaction.

²¹ This way we avoid problems with varying boolean type support in different databases and database tools.

²² This way we avoid typing errors which could occur if we used constant string values.

²³ These tools allow transformations between Hibernate configuration files, database schemas and (annotated) Java source code. We use the automatic creation of Java data object classes and the automatic creation of database tables. Regarding the data base, additional ANT targets are used which contain SQL statements to create additional database entries like (example) master data and views.

²⁴ Pessimistic locking is realized by setting lock mode `UPGRADE` for Hibernate queries. Depending on the database used, Hibernate automatically uses the right mechanism to ensure that queried data cannot be changed until the transaction is finished. The table `T_DB_SEMAPHORE` is updated every time a transaction is started in order to lock access to other tables.

A utility class *HibernateSessions* provides static methods which should be used for all transaction handling.

General scheme

```
session = HibernateSessions.startTransaction (getClass ());
try {
    ...
} catch (RuntimeException e) {
    HibernateSessions.rollbackTransaction (session, getClass ());
    throw e;
} finally {
    HibernateSessions.finishTransaction (session, getClass ());
}
```

5.17.1 Transactions and User Interaction

The strategy of short transactions determines automatically that no transaction may stay open during user input. Therefore, for web applications we use the *session-per-request* pattern. This means that an HTTP request is processed within exactly one transaction.

Typical user interaction patterns like *view-edit-save* are therefore implemented within several transactions. Thus, the transaction concept does not automatically prevent e.g. parallel modification of data by several users. It is the responsibility of the application to handle such situations.²⁵

²⁵ Two strategies are possible:

Last save wins

This can be a practical solution if parallel modifications are unlikely, e.g. for data which may only be modified by a small group of administrators.

Recognition of inconsistencies

Hibernate contains support for the recognition of stale data (using e.g. automatic version numbering of table entries). Thus, the user could be provided with an error message to prevent modification.

Currently we always use the first strategy. Note that because of referential integrity checks by the database, there is no danger that this strategy might lead to inconsistent data.

5.17.2 *Commit and Rollback*

Hibernate sessions are always created at the highest possible level, e.g. within the Struts action classes. Therefore, they are also closed and committed at this level—usually within a *finally* block of an action method.

Conforming to our exception strategy we also catch Evolaris exceptions at the highest possible level. In order to ensure that no invalid data remains in the database, it is always necessary to trigger a rollback within the *catch* blocks. Hibernate sessions are thus usually opened and closed at the same level as exceptions catching, i.e. within the top level of web interface or EJB code, because a rollback has to be performed within the catch clause.

5.17.3 *Locking*

The platform uses logical locking across tables. Thus, not the individual tables are locked, but, with the help of a special semaphore table, all entries belonging to specific groups or even all entries of all groups are marked as locked.

Group-Specific Locking

Users of the platform administrative interface without administrator role may only access data of their own groups. Also, modelled services within interaction list processing, caused for example by incoming messages, timer events or scanner redemptions, may only access data of the group the interaction list is assigned to. In these cases it is sufficient to lock only data from the affected groups. Then, several platform users and services can be processed in parallel and there is no danger that erroneous code accessing data of a specific group also locks access to data of other groups.

Normally, the platform automatically starts transactions with correct group locking within the provided Struts base actions and when preparing interaction list processing. However, in special cases, e.g. when long-lasting operations have to be performed within several short transactions,

also the programmer of client project implementations has to care about correct locking. Extended methods within the *HibernateSession* class allow passing a group ID parameter to specify that only data of a single group shall be locked. As an example, the following statements wrap code which only accesses and locks data of the group with ID 7:

```
session = HibernateSessions.startTransaction (getClass (), 7, null);
    try {
        ...
    } catch (RuntimeException e) {
        HibernateSessions.rollbackTransaction (session, getClass ());
        throw e;
    } finally {
        HibernateSessions.finishTransaction (session, getClass ());
    }
```

Global Locking

If data of more than one group or group-independent data is accessed within a session, access to all entries of all groups has to be locked during the transaction. This is performed automatically by the platform when users with the administrator role access administrative platform web pages. Framework code accessing group-independent data also uses global locking.

Global locking is performed by class *HibernateSessions* when no group ID or value null is passed to the start transaction method.

5.18 Referential Integrity

Referential integrity is used between tables containing interrelated data which must not be separated. Deleting an entry in a table is only possible if all entries in other tables linked with referential integrity are also

deleted.²⁶ It is therefore usually not possible to remove entries which are referenced by other tables. Other mechanisms like removing a user from a user set shall be used instead of deleting.

5.18.1 Automatic Creation of Restrictions

Referential integrity restrictions are automatically created by Hibernate for all *many-to-one* associations. A group reference within a user table can be defined like this:

```
<many-to-one name="group" class="evolaris.usermanagement.datamodel.Group"
fetch="select" foreign-key="FK_USERS_GROUPS"
not-null="true">
    <column name="FK_GROUP_ID" precision="38" scale="0" not-
null="true"/>
</many-to-one>
```

If the user table is created with the Hibernate tools, the created restriction will make sure that no group can be deleted which is referenced by at least one user table entry. In this particular case, this would never be a problem, because we do not allow deleting group entries within applications.

5.19 Data Layer Artifacts

5.19.1 Framework Database Project

The database project contains all framework data layer components including Hibernate mapping and configuration files, Java data objects and scripts for database setup. Additionally, some tables needed by client projects without own data layer configurations are contained there. These should be eliminated in the future.

All framework code and client-project-specific code accesses framework data via objects within the database project.

²⁶ It would not make sense, for example, to keep game preferences for a user after he has been removed from the system, because they could never be used in a correct context again. Optionally, the application may require a confirmation by the user before deleting or it may require the user to explicitly delete dependent data.

Although we use a single Eclipse project for all framework data layer code, the package paths and build scripts are subdivided according to the modules they belong to.

5.19.2 *Client-Projects with Data Layer Artifacts*

Client projects which are configured and deployed independently may use additional client-project-specific tables which themselves refer to framework tables within the database project. No separate Eclipse projects are required for this, usually the main business layer project is used to contain Hibernate files, Java data objects and build scripts for these additional tables.

A Hibernate configuration file containing references to the additional tables has to be configured within table *T_SYS_CLIENT_PROJECTS*, so the framework knows which additional tables to access when starting client project actions and services.

5.20 Extending the Database

Completely rebuilding the database is only possible on developer PCs and test servers. On the production server, existing data has to be preserved and it might be necessary to avoid changes interfering with existing deployed applications.

5.20.1 *Restrictions*

Using a single database schema and instance for framework and client-project data has got important consequences concerning compatibility and extensibility. Particularly, unless all deployed EARs are updated immediately after performing structural changes in the database, no tables may be removed, existing tables may only be extended in a backwards compatible way.²⁷ Otherwise, running applications would crash.

²⁷ This way we make sure that the following requirements are fulfilled:

- Applications within enterprise archives can be deployed and undeployed independently.

5.20.2 Rules

Based on these restrictions, *only* the following changes to the database schema are allowed:

- Adding new attributes to existing tables, with either a default value provided or *null* values allowed (no *not-null* setting in the Hibernate mapping file)
- Adding new tables and views which may also refer to existing tables and views (but not vice versa)
- Changing the names of Java data object properties, as long as the database attribute name stays the same.
- Removing attributes in the mapping file as long as the attribute remains in the production database; the attribute has to allow null values, otherwise a default value is required. In this case a comment "DEPRECATED" should be added to the table with a database administration tool.

Within Java code also the following restriction has to be obeyed in order to remain compatible:

- Only empty constructors of data access classes (POJOs) created by Hibernate tools may be used, because the interfaces of the other constructors change every time a required attribute is added to a table.

-
- Changes in the database must not require changing or even redeploying running applications.
 - Existing data must not be affected by structural change