Hildeberto Mendonca

UGM - User Group Management

An Open Source Application that Implements the User Group Domain

January 17, 2013

CEJUG - The Ceara Java User Group

Contents

List of Figures

List of Tables

1

Introduction

1.1 What is a User Group?

1.2 Case Study

The chosen case study is based on the personal experience of the team who is developing UGM. Most of contributions come from the Ceará Java User Group, which is a technical user group located in the Northeast cost of Brazil. Despite being part of a specific domain, we believe that the proposed model is generic enough to cover the management's needs of most user groups out there.

1.2.1 The Java User Group Scenario

Java User Groups (JUG) are independent and entrepreneur communities of students and professionals around the Java platform and related technologies. They are globally spread with larger concentration in Europe, North America and South America.

Their independence from the industry is particularly interesting because, in general, user groups are promoted by the industry, which is the case of GUG, attached to Google; OUG, attached to Oracle; and MUG, attached to Microsoft.

Part I

User Guide

2

Membership Management

2.1 Registration

The decision to register in the user group always come from the interested person. That is why the only way to add a new member is filling out the initial registration form, accessible through a link in the application header when there is no member logged in. JUG leaders do not have any feature that allow them to add new members manually.

Besides the basic personal data, the registration form also asks the interested person to select one or more of the following options:

- "I want to be aware of all events organized and supported by the UG in a local, national and international level" - as a member, the person will receive information about events organized by the UG (see Chapter 3) and other supported events promoted by partners and sponsors.
- "I want to receive sponsors' and supporters' offers of products and services, such as books, courses, magazines, etc" - only those members who checked this option will be able to participate in raffles, promotions and other contests promoted by sponsors.
- "I want to receive local and national job offers related to Java" - only those members who checked this option will receive job offers from partners and sponsors.
- "I want to participate in the technical discussion list" - checking this option, the member will be automatically registered in the technical mailing list.
- "I want to receive news about Java and other related technologies, as well as news about the market and other communities" - only those members who checked this option will receive news about subjects discussed in the group and other community activities.
- "Other UG members will be able to see my profile and contact me directly through the application. The application will carefully protect the email

address, not showing it for others" - only members who checked this option will be able to use the social features of the application.

When the interested person submits the registration form, we have to make sure that his/her email address is correct before considering him/her as a member. We send an email message to the email address informed in the form, asking the interested person to confirm their email address by clicking on the confirmation link. This link contains a unique code that guarantees that the link cannot be reused after its first use, confirming the email address only once. The interested person is considered as a member as soon as his/her email address is confirmed. The new member receives a welcome email message and JUG Leaders are informed by email about the successful member registration.

2.2 Member Profile

The member has the right to read and modify any data published on its own profile. The email address is the only data subject of validation. The email validation works the same way it works during the registration: the system sends a message to the new email address with a confirmation code to validate it.

2.3 Deactivation

The deactivation of a member means that he/she will not participate in the activities of the group starting from the date of the deactivation. No email message, invitation, offer, or any other kind of information will be sent to the deactivated member any longer. Therefore, he/she is not considered as a regular member.

At the same time, all data input by the ancient member in the database will not be removed. Comments, email messages, articles and other information will be kept unchanged indefinitely. Therefore, any modification on these data is not responsibility of the application.

3

Event Management

Events are strategic for user groups and for companies that want to promote the use of their technology or attract talent people to compose their dream team. The occasion is appropriate to disseminate knowledge and promote networking, strengthening the links between people and increasing the likelihood of new opportunities. Nowadays, there are several ways to build a network virtually, but the body language still is the most efficient way to know people's behaviour, reliability and friendliness. Those good experiences Good ex

It is important to have an efficient event management in order to keep everything under control, measure member's participation and get their feedback.

3.1 Event Registration

To start organizing events, UG Leaders should register venues available and suitable for user group events. A venue might have one or more rooms where event sessions will take place. In case of multiple rooms available, the event can manage multiple sessions in parallel.

Events are allocated in existing venues for a certain time. When an event is registered and allocated to a venue, the venue's contact receives a request email containing details about the event and a list of resources that are expected from them. After a negotiation process the event is confirmed or not. The confirmation occurs when the venue's contact clicks on the confirmation link in the email message. Without this confirmation the event cannot occur.

For each session of the event, one or more speakers may be allocated. A speaker is a qualified person on the subject(s) of the session. He/she is invited by a UG Leader to give a speech, training, coordinate a discussion or any other social activity. The person is invited to register in the application as a speaker. Once logged in, he/she can put a short summary of his/her experience, an abstract of the session, upload his/her profile picture, presentations,

documents, source code, links, and other useful contents for the session. The application will use all this information to compose the page of the event.

When the event is confirmed, an email message containing detailed information is sent to all members that have declared in their registration form the wish to receive information about events. This message contains a direct link to the event registration form.

Detailed information about the event is also formatted to be published on the UG website. Consequently, it may attract people who are interested on the event but are not member of the UG. Those people should become a member of the UG before registering to the event.

Right after the event registration, the member receives a confirmation message just to let him/her know that he/she is successfully registered to attend the event. Registered members will receive a remind email message seven days before the event and a second one on the day before the event. They can cancel their registration at any time before the event.

3.2 Controlling the Event

At the entrance of the event, a member of the UG staff checks the inscription of each person in an available computer. If the person is a member registered in the event, then his/her presence is confirmed. If the person is a member but he/she is not registered in the event, then his/her registration is made at the entrance. If the person is not a member, then he/she should agree to become a member of the UG, otherwise it is not possible to join the event. If he/she agrees to become a member, his/her registration in the UG and in the event is done at the entrance of the event.

3.3 After the Event

When the event has passed, no information about the event can be changed anymore. It will be available only in a read-only mode from the day after. Speakers will not be able to update their profiles either until being allocated to another event. However, UG Leaders are able to publish additional resources related to the event, such as pictures, documents, presentations, etc.

4

Partnership Management

4.1 Registration of Partners

4.2 Sponsorship

4.3 Exclusive Services for Partners

4.3.1 Dissemination of Job Offers

4.3.2 Offers of Products and Services

4.3.3 Search for Talent People

5

Knowledge Management

5.1 Topics in the Scope of the User Group

5.2 Mailing Lists Management

Part II

Development Guide

6

Architecture

UGM should be deployed in an Application Server (AS). This platform is responsible for the system execution, the connectivity to several resources available in the environment and the availability for several users and systems. For the moment, Glassfish 3.0.1 is the standard AS for this application. This JEE Server has gained lots of attention for constantly run towards the state-of-the-art of the Java server side technology. It has an aggressive roadmap, being always the first product on the market to fully implement the last version of the JEE specification.

One of the main advantages of using an application server like Glassfish is to keep the application free of complex code, such as a) manual control of database transactions; b) database access configuration; c) security authentication and authorization; e) sending and receiving e-mail messages, among many other complexities that are non-functional requirements, consuming the time we would be spending on functional requirements.

No test was performed in other AS up to now, mainly because when the project started Glassfish was the only AS implementing the JEE 6 specification (JPA2, EJB 3.1, JSF 2.0, etc.). The AS should have access and provide the following resources:

- *Security Realm*: is a provider of authentication and authorization data to identify users who want to access the application and verify whether they have rights to access protected resources. The application's database, besides storing UG's data, also stores users and their roles, which makes it the realm provider, referenced by Glassfish in the realm configuration.
- *Database Server*: a database driver allows connections to a relational database and a connection pool manages several simultaneous connections to this database, allowing scalability and performance. MySQL is the database system chosen to organize and protect UG's data.
- *Email Server*: A JavaMail service connects to an email server and provide sessions to the application, allowing it to send and receive email messages without any special configuration on the implementation side.

- *File System*: the application should be able to save and access files in a specific directory of the file system. Unfortunately, the AS does not manage access to the file system. The application is responsible for managing all uploaded files.

The resources provided by the application server help to simplify the overall application architecture. The connectivity with those resources characterizes a n-tier architecture, where each tier is physically separated. The application architecture, in turn, is divided in four layers, as depicted in Figure 1. These layers provide a model to create a flexible and reusable implementation. This way, new features can be efficiently accommodated, with a minimal impact on existing code.

The 4 logical layers are:

- *View*: implements the user interface by defining the layout of the screens, position UI components, performing data validation (e.g. invalid date format), formatting data to be presented, internationalizing and localizing text, and giving feedback about the user interaction.
- *Controller*: controls the navigation flow according to the user interaction and intermediates data from the view to the business and vice-versa. It performs business validation (e.g. user already exists), data conversion from user friendly to business model-compliant and vice-versa, accesses business services, creates model objects that do not exist yet and manipulates existing ones. According to the state of the IS, the controller knows which flow should be followed and what to do in case of exceptions and security constraints.
- *Business*: due to its objectivity, the business layer is not subdivided. It is considered as concrete and it will perform transactional business operations over the data model. Every business operation should guarantee that the data model is consistent before and after its execution. Therefore, the data model can only be access through this layer.
- *Persistence*: maps data model entities with database tables to manage the lifecycle of entity objects. These objects can be created (insert), updated (update), queried (select) and deleted (delete). These operations are widely used by the data access object layer in order to interact with the database. The persistence layer can manage one or more data sources, but a data source is managed by one and only one data source.

6.1 Chosen Technologies

Table 6.1 lists the technologies adopted by the development team to implement the application.

These technologies were selected based on the current needs, resources and knowledge of the development team. We decided to adopt a minimalist approach where most libs needed by the application are also distributed

Table 6.1. Chosen technologies to implement logical layers

Logical Layer	Technology	Version
View	JSF Facelets	2.0
	Primefaces UI Library	2.1.1
Controller	Primefaces	2.1.1
	JSF Managed Beans	2.0
Business	EJB Session Beans	3.1
	EJB Timer	3.1
Persistence	JPA	2.0
	EclipseLink	2.0
	JTA	1.2

with the AS, such as Eclipse Link and Mojarra, and most configurations are made through the administrative console, such as database connection pool, JavaMail session and Security Realm. For the moment, the only external lib is the Primefaces component library. We make extensive use of annotations and avoid as much as we can XML for configuration purpose. Transactions are fully managed by the container. This way, we keep focused on the source code of the UG community model. Eventually, other technologies out of this table may be adopted if well justified. Therefore, a new technology would be considered in case a very special UG's need must be fulfilled.

Going into details about the chosen technologies, we describe each one of them, completing the description of Figure 1:

- **JSF 2.0**: The Java Server Faces technology is the standard technology for developing web-based applications in the JEE platform.
  - *Primefaces*: extensive library of UI Widgets available for the JSF technology.
  - *Converters*: converts data from user friendly to business model-compliant and vice-versa.
  - *ManagedBeans*: POJO annotated classes that have access to special resources available on the application context.
  - *Validators*: performs server-side validation of data informed by the user before going forward in the controller processing.
- **JNDI**: the Java Naming and Directory Service helps to localize and retrieve instances of resources available in the context of the server, reusing existing instances and avoiding the complexity behind creating those instances.
- **EJB 3.1**: transactional, distributed and secure component model to encapsulate reusable business logic.
  - *Stateless Session Beans*: EJB that does not store the state of components in memory, optimizing memory allocation and scalability across multiple servers.

  - *Timer*: EJB capable of scheduling itself to execute business logic in a certain time or in a certain frequency of time. The schedule of routines is very appropriate to perform automatic maintenance tasks such as cleaning temporary data, generating complex reports, sending alert messages, etc. Timers are also useful to efficiently use computational resources when systems are in idle mode.
  - *ManagedBeans*: POJO annotated classes that have access to special resources available on the enterprise context.
- **JPA 2.0**: Java Persistence API is an entity-relational mapping specification that manages the lifecycle of objects persisted in the database. It reduces the degree of database dependency, the complexity of the source code and the maintenance cost in case of changes in the relational model.
  - *Entity Model*: POJO annotated classes mapped to database tables where their instances represent table records for the business logic.
  - *JTA*: The Java Transaction API.

6.2 Database Model

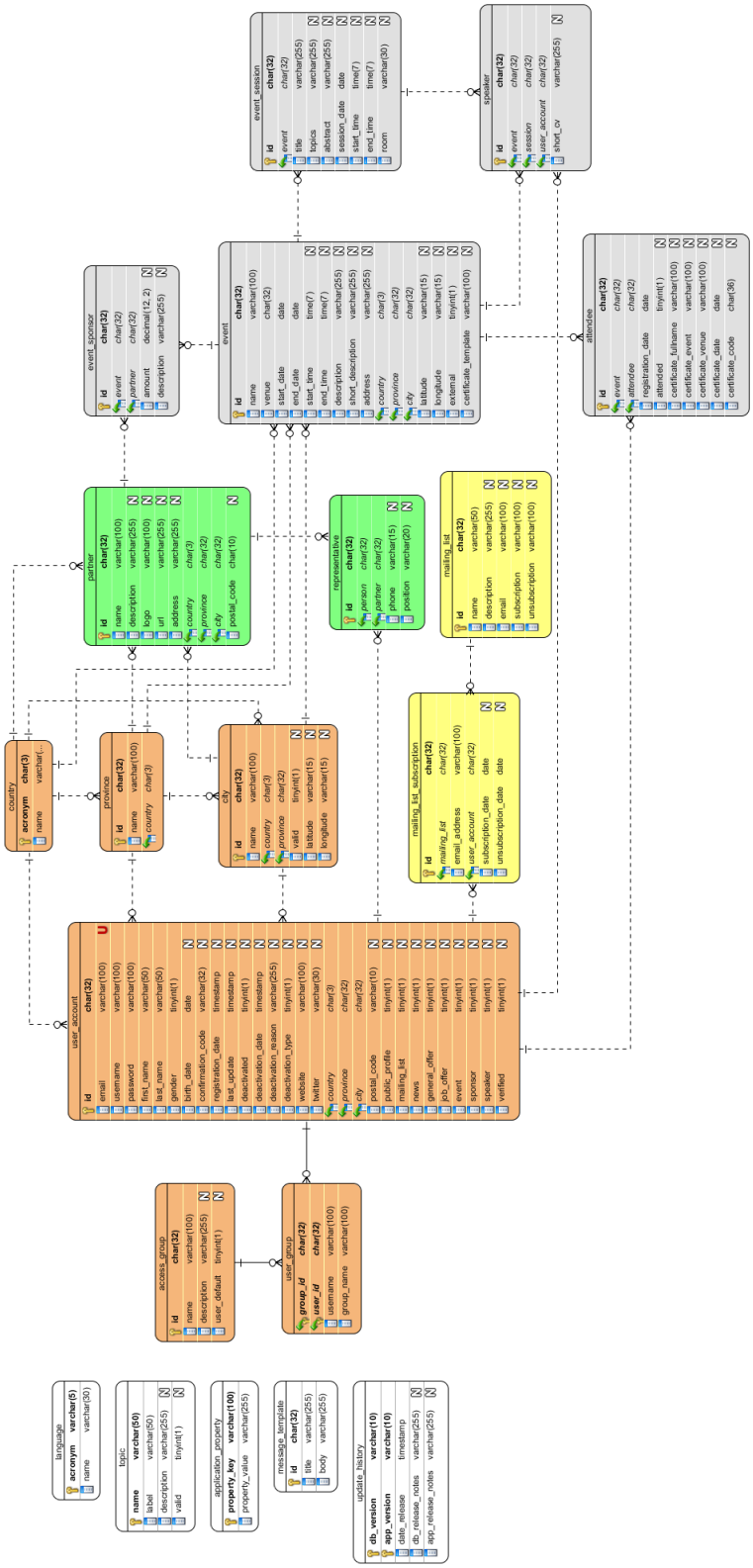

Fig. 6.1. Please write your figure caption here

7

Installation

To install UGM we need:

- Java Standard Edition JDK 6;
- Glassfish 3.0.1 Server or superior;
- MySQL 5 or superior;
- MySQL ConnectorJ JDBC Driver;
- Access to a SMTP and POP 3 server to send and receive email messages.

The initial installation takes some time, but it will guarantee easy updates when new releases come out. Because the installation of Java SE JDK, Glassfish and MySQL are dependent of platform, we consider that they were already performed by the administrator, who knows details about the hosting system. We would not be complete and up-to-date enough here, thus the best source of information is their respective websites. The steps to install the application are:

1. Create a MySQL database: run SQL scripts to create a database and its structure dedicated for the application. The last available version of the database script is mysql-create.sql.
2. Install the JDBC Driver on Glassfish: make the MySQL ConnectorJ JDBC Driver available in the application server classpath to be used by the connection pool to create new MySQL database connections.
3. Create a database connection pool on Glassfish: the connection pool manages connections to the database using the JDBC Driver.
4. Create a datasource for the connection pool on Glassfish: the datasource links the connection pool to the application. In other words, the connection pool is a resource and the datasource is a name for this resource. This name is used in the application to locate and use the resource.
5. Create a Security Realm: the security realm allows the declarative implementation of the application's security, significantly reducing the application complexity by delegating this responsibility to the container.

6. Create a JavaMail session: since this application deals with people, it has to send emails very often. Therefore, the availability of a JavaMail session, managed by the container, is essential to support the high demand for sending and receiving emails without dealing with the complexity of managing email server connections.
7. Deploy the application package: finally, we deploy the application package that makes use of all configurations above.
8. Configure the application according to specific needs of the JUG: after the initial deployment the application will run normally using default configuration. However, the application will be fully operational only when specific configurations for the JUG are defined.

7.1 Creating the MySQL Database

UGM has access to only one database. This database is created using the administrative console and a SQL script. As mentioned before, this instructions consider that the MySQL database, version 5.0 or superior, is already installed and configured. In terms of configuration, we also consider that the path of the operating system is pointing to the folder where all MySQL commands are located.

The procedure starts executing the administrative console, using the following command:

```
# mysql -u root -p
```

Usually, an administrative user is needed to create a new database. The user *root* is the one with privileges to perform this operation. It will create a client authenticated session to access MySQL. *-u* means that the subsequent value is the user of the session and *-p* means that the password should be prompted right after executing the command. Once authenticated, the user *root* will allow the execution of the command below, which creates the database and a dedicated user for it:

```
mysql> create database jug;
mysql> create user jug\_user identified by '[password]';
mysql> use jug;
mysql> grant all privileges on jug.* to jug\_user@'%';
mysql> flush privileges;
```

The database *jug* and the user *jug_user* are created, and all privileges over the *jug* database are granted to the *jug_user*. To check if the database was created, execute the following command:

```
mysql> show databases;
```

Check if the database *jug* is in the list. Then we have to sign out from this root's session and open a new session for the new user to create the database structure. Follow the sequence of commands below:

```
mysql> quit;

# mysql -u jug_user -p

mysql> use jug;
mysql> source [path of the database script]/mysql_create.sql;
mysql> quit;
```

The user *jug_user* and its password should be used in the configuration of the connection pool. Do not use the *root* user for application purposes.

7.2 Installing the JDBC Driver on Glassfish

The driver installation consists on saving the driver file in a specific AS lib directory. For this step instructions consider that Glassfish AS, version 3.0 or superior, is already installed and configured. The driver is available to download at MySQL's website (http://www.mysql.com). The downloaded file contains the driver and its documentation. Copy only the driver file, mysql-connector-java-[version]-bin.jar, to the diretory [glassfish_home]/glassfish/domains/domain1/lib/. Restart the AS to make the driver available.

7.3 Creating a Database Connection Pool

The database connection pool is managed by the AS, thus it is configured in the administrative console. The necessary steps are:

1. Enter in the administrative console at http://[servername]:4848/ and navigate to Resources / JDBC / Connection Pools.
2. Create a new connection pool with the name jugPool, select the resource type javax.sql.ConnectionPoolDataSource, select the database vendor MySQL and click next.
3. Select the datasource classname *com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource* and inform the following additional properties:
   a) DatabaseName=jug
   b) User=jug_user
   c) Password=[password]
   d) PortNumber=3306 (this is the default port but make sure that you are using the correct one)
   e) ServerName=[server-name or ip]

4. Click on *Finish* to save the new connection pool.
5. Go to the list of connection pools again and select the new one that was just created.
6. Click on *Ping* to check if the connection was correctly configured. The message "Ping Succeeded" means that the connection is working fine.

7.4 Creating a Data Source for the Connection Pool

To be able to use this connection pool in JEE applications, we have to create a JNDI name for it. This configuration is also done in the administrative console, at *Resources/JDBC/JDBC Resources*.

Click on *New* to start the creation of the data source. Set the field *JNDI Name* to *jdbc/jug*, select the connection pool *jugPool* (see section 7.3) and click *Ok* to finish. This JNDI name will be used by the application to access MySQL database.

7.5 Creating a Security Realm

As the connection pool and the JavaMail session, the security realm is also a configuration entirely made in the application server. Users' data are stored in the database, thus this is a JDBC Realm, which uses the data source created in section 7.4. Follow the steps below to configure the JDBC Realm:

1. enter in the administrative console (http://[server-name]:4848).
2. Go to Configuration / Security / Realms, press "New..." and inform the following values:
   a) Name: jug-realm
   b) Class Name: com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm
   c) JAAS Context: jdbcRealm
   d) JNDI: jdbc/jug - the data source JNDI name pointing to the connection pool where user data are available
   e) User Table: user_account
   f) User Name Column: username
   g) Password Column: password
   h) Group Table: user_group
   i) Group Name Column: group_name
   j) Digest Algorithm: MD5
   k) Encoding: Base64
3. Restart the application server to activate the realm.

7.6 Creating a JavaMail Session

Besides managing the connections to the email server, the JavaMail session also stores security parameters, such as user name and password, besides other details such as SSL and the kind of server available. These parameters are defined in a case by case basis, thus the JavaMail service provided by the application server is essential for the distribution of this application. To configure JavaMail, follow the steps below:

1. enter in the administrative console (http://[server-name]:4848/).
2. go to *Resources / JavaMail Sessions*.
3. create a new JavaMail session and set the following properties:
   a) JNDI Name: mail/jug
   b) Mail Host: [smtp-server-address]
   c) Default User: the username to authenticate on the smtp server
   d) Default Return Address: the address used by recipients to reply the message. Some servers require that this address should be the one used by the authenticated user to access his mailbox.

If the server doesn't request secure authentication, then the three steps above are enough to start using the new JavaMail session, but a server without secure authentication is a very rare case nowadays. Therefore, we will certainly need to inform a password to login on the smtp server. In most cases, the server administrator also changes the default port of the smtp server, which forces us to explicitly inform the correct port. For these special needs we can use additional properties in the JavaMail session. Follow the steps below:

1. Still on the JavaMail session form, go to the *Additional Properties* section and add 3 more properties, which are:
   a) mail.smtp.port: [port-number]
   b) mail.smtp.auth: true
   c) mail.smtp.password: [password]
2. Click on Save to create the JavaMail session.

7.7 Deploying the Application Package

The installation process takes some time, but all these steps are executed only once, simplifying significantly the development process by going from development to test and production without any specific application customization. The deployment process is also simplified, reducing this final phase, as follows:

1. enter in the administrative console (http://[server-name]:4848).
2. Go to *Applications* and press *Deploy...*.
3. Inform the package *jug.ear* to be uploaded to the server.
4. Select type *Enterprise Application*.

5. Click *Ok* to deploy.
6. Check if everything went well accessing http://localhost:8080/jug if deploying on your own computer or http://[servername]:8080/jug if deploying on the server.

UGM will appear in the list of applications. In case of updating an application already deployed, the process is very simplified because of the initial configuration:

1. enter in the administrative console (http://[server-name]:4848).
2. Go to *Applications*.
3. click on the action *Redeploy* in the row of the jug application;
4. inform the package jug.ear to be uploaded to the server; and
5. click *Ok* to redeploy.
6. Check if everything went well accessing http://[server-name]:8080/jug.

7.8 Configuring the Application According to the UG's Specific Needs

Before using the application, the database should be initialized with some initial data that are specific for each UG. The following steps are necessary:

1. registration of the administrative user: after the initial deployment, there is no user registered in the application. Even the administrator is not present. When the user tries to authenticate while there is no registered user, the navigation redirects him/her to the registration page. On the top of the form, the user is alerted that he/she is going to be the first member and will be considered as administrator. The form is a simplified version of the original registration form because some mandatory data are not available yet, such as geographic location. As an initial user, he/she is automatically confirmed, without the need of a confirmation by email.
2. definition of the UG location and geographic coverage: because country and city are mandatory fields in the registration form for all other users, the administrator should define them as soon as he/she have access to the application. The administrator should register at least a country to allow the registration of members, but it is expected from him/her that the location of the UG and its coverage are also registered. It means that the administrator should register all countries, provinces and cities where the UG has influence.
3. definition of application properties: at last, but not least, the administrator should review and modify, if needed, the default value of application properties. This properties are initialized in a way that allows the application run normally. However, some properties are missing by default and others might be different from what is expected.

7.9 Application Update

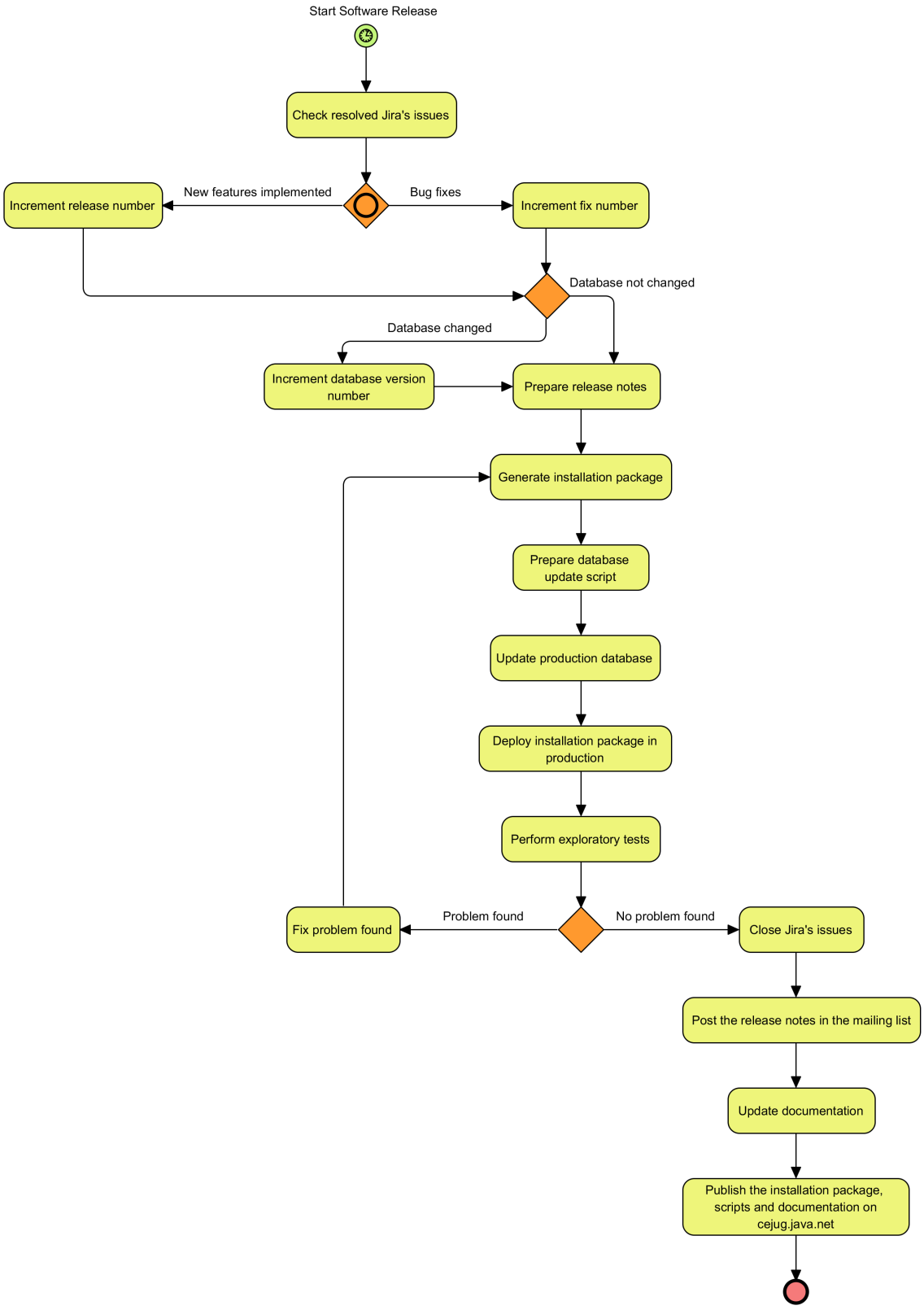

Fig. 7.1. Release process

8

Development

8.1 Preparing the Development Environment

Git is the version control system (VCS) used to control UGM source code. For being a distributed VCS, Git is not trivial. Its foundations are a bit different from traditional VCSs, such as CVS and SVN. There is a master repository located at http://cejug.java.net, where the main trunk (master) is centralized, but every client is also a full featured VCS, where is possible perform commits off line, create branches and merge with other clients without acknowledging the master.

To install and use UGM is not necessary to start from the source code. The binary package is available at http://cejug.java.net. Just follow the instructions described in chapter 7. On the other hand, if the intention is to contribute to the project, the use of Git to retrieve the source code is necessary. The potential contributor should:

1. install Git in the development machine: the installation is dependent of platform, thus the instructions for the target platform are only available on the Git website: http://git-scm.com.
2. configure Git to access the master repository: UGM's Git accepts only secure SSH (Security Shell protocol) connection, making the initial steps more complicated than expected. After this initial configuration, the access to the master becomes transparent and secure.
3. register at java.net:
4. copy and paste the public key in the java.net personal profile:
5. register yourself in CEJUG's project at java.net. You will have access to the repository only if you are a member of the project.
6. When you login on Java.net and go to the source code section, you will be able to see the checkout URI of UGM's repository. The address contains your username. Example: ssh://[user-name]@git.java.net/cejug jug-management.

7. Copy the URI above and go to the command line and type the following command to checkout the code: git clone ssh://[user-name]@git.java.net/cejug jug-management. It will create a folder "cejug-jug-management" in the directory you ran the command and the full source code will be available there.

To make sure that the code in the server branch is perfectly working, we have to update the local copy with the latest changes available on the server, test the system locally with those changes and if everything is working well, the local commits can finally be pushed to its respective branch. Before updating the local, make sure that there is no pending commit using:

```
# git status
```

If there are changed files add them using:

```
# git add [file]
```

Use the command above for each modified file or if they all make sense to be committed together use:

```
# git commit -a -m [message]
```

to add all changed files and commit right after. If files were added individually use:

```
# git commit -m [message]
```

After performing the last commits, it is time to update the local copy using:

```
# git pull
```

or

```
# git fetch
# git merge FETCH_HEAD
```

To push local commit to the server, use:

```
#git push origin master
```