Nibiru 0.4 Reference

January 31, 2013

http://nibiru.googlecode.com

Part I

Introduction

1 Framework objective

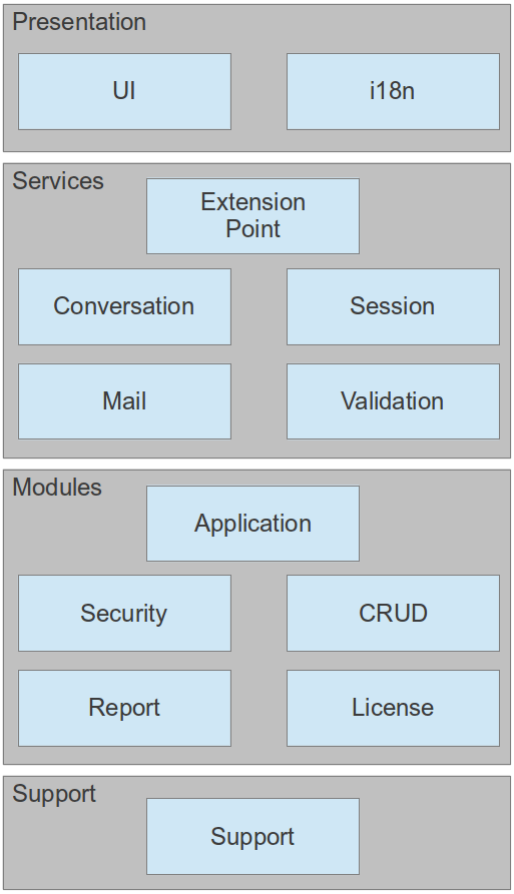
The framework objective is to facilitate the building of modular applications. The following goals are established in order to meet such objective:

- Providing an abstraction layer over different technologies in order to avoid coupling.
- Providing services which are common to business applications, such as CRUDs, reports, workflow, transaction management, security and internationalization.
- Providing dynamic update mechanisms for the system in order to allow hot swapping.
- Implementing patterns which facilitate solving problems in a structured way. But avoiding to force the user to implement a given solution.
- Facilitate decoupled communication among modules.
- Avoiding reinvent the wheel. Creating layers of abstraction butusing existing technologies when possible.

2 Architecture

This section explains architectural concepts and decisions.

2.1 High-level diagram



2.2 IoC pattern

In order to decouple each component from the container and other components, the dependencies of each component are injected (IoC pattern).

2.3 MVP pattern

The model used for the presentation layer is the MVP pattern, under its passive view variant. This allows the presenters to be decoupled from each other by an

event bus and also to be decoupled from view implementation. Google also makes a good description of this pattern.

Also, the concept of abstracting the view was taken a step further, creating abstractions for common components. Thus, the user can choose creating a generic view or creating a view using the particular advantages of a specific technology.

2.4 Extension points

The system has an extension point mechanism for adding or removing functionality dynamically. The idea was taken from Eclipse platform, but trying to take a simpler approach.

2.5 Java platform

Javawas chosen because it is currently the most widespread platform within the enterprise applications, in addition to being easily portable to different environments and having many frameworks and libraries.

2.6 OSGi / Blueprint

We chose OSGi because it provides a mechanism for dynamic module management. Blueprint is used because it provides many facilities to implement the IoC pattern under OSGi.

Using these technologies, shared components are exposed using OSGi services. Also, the division between API and implementation allows service hot swapping, since the client components doesn't access to the concrete class implementation. On the other hand, Blueprint provides proxies that make such hot swapping transparent to the client code.

However, almost all components are independent of OSGi and Blueprint, thanks to the IoC pattern (except for the ones that implement specific OSGi features - which can be replaced). This way, we provide support for non-OSGi deployment too.

3 Getting started

3.1 Required software

1. Java (http://www.java.com/en/download/).
2. Eclipse (http://www.eclipse.org/).

3. Maven (http://maven.apache.org/).
4. A GIT client (http://git-scm.com/). We use EGit.

3.2 Installation

1. Clone the project as explained in http://code.google.com/p/nibiru/source/checkout
2. Run "mvn eclipse:eclipse" from root directory in order to build the Eclipse project from Maven files and downloading target platform JARS.
3. A target platform, with all the dependencies, will be created in ar.com.oxen.sample/ar.com.oxen.sample.ta If not (or if you change the dependencies) go to that project and run "mvn compile" in order to rebuild the target platform from Maven dependencies.
4. Import the projects into Eclipse. You must create a M2_REPO classpath variable pointing to the m2/repository directory in your home directory.
5. At preferences menu, activate the "Nibiru Sample" target platform. Select "reload" option in order to recognize the downloaded JARs.
6. Run the OSGi application launch called "Nibiru Sample". By default, Eclipse adds all the plugin (OSGi) projects which are open in the workspace, so even if there is a JAR with the same module, the source project takes precedence.

You can run the sample inside a non-OSGi environment. The ar.com.oxen.nibiru.sample.springwebapp project does this. It runs as an standard WAR into a servlet container. Download its binaries from here.

3.3 Sample project

Running the sample application will create an H2 database in a directory called nibiruDb inside your home directory. Windows users should modify the ar.com.oxen.nibiru.sample/ar.com.oxen.nibiru.sample.datasource.fragment/src/main/resources/database.prop file in order to specify the database location.

The sample application uses a dummy authentication service. Log-in with user "guest", password "guest".

TODO: Simplificar el armado de un proyecto. Opciones:

1. *Hacer un namespace handler.*
2. *Armar anotaciones.*
3. *Usar directamente Guice+Peaberry (http://code.google.com/p/peaberry/)*

4. *Crear un DSL con Builders*

Part II

Project Structure

4 Main subprojects

The structure for Nibiru project is arranged in an hierarchical way. In this structure, the bundles are arranged into the following main groups:

- ar.com.oxen.nibiru.application
- ar.com.oxen.nibiru.conversation
- ar.com.oxen.nibiru.crud
- ar.com.oxen.nibiru.extensionpoint
- ar.com.oxen.nibiru.i18n
- ar.com.oxen.nibiru.license
- ar.com.oxen.nibiru.mail
- ar.com.oxen.nibiru.report
- ar.com.oxen.nibiru.security
- ar.com.oxen.nibiru.session
- ar.com.oxen.nibiru.support
- ar.com.oxen.nibiru.ui
- ar.com.oxen.nibiru.validation

They can be found on a directory called “main”

5 Sample project

A sample application can be found in the ar.com.oxen.nibiru.sample project. It can be found at “sample” directory.

6 Categorization

6.1 API / implementation separation

We define two kind of modules, in order to facilitate the decoupling among different modules implementations:

- API: Contains interfaces to be exposed to other components. By convention the name ends with ".api".
- Implementation: Contains API implementations. By convention the names are almost equals to the implemented API name, but changing ".api" suffix by something descriptive of the implementation.

In general, any module can only access another module through an API. The exception to this rule are modules with utility classes that do not expose services.

Another naming convention is that implementations of APIs that are not dependent on a particular technology will have a ".generic" suffix.

6.2 Division between classes and services

XML for exposing class instances as a services is stored in a separated bundle. This way, you can export services in a custom way simply by installing a different bundle, with a custom XML configuration (or even using another technology, such as Peaberry). An you can reuse the classes from the main bundle as you want.

Bundles with service exporting definitions are suffixed with “.service” on the name.

Part III

Modules

7 Base application

The ar.com.oxen.nibiru.application.api bundle contains interfaces used to implement basic functions such as application login, "about" window , etc.

The idea is that an implementation of this bundle must provide the basis to setput the application. All the extra functionality will be added by other modules.

This module contains factories for presenters:

```
package ar.com.oxen.nibiru.application.api;

import ar.com.oxen.nibiru.application.api.about.AboutView;
import ar.com.oxen.nibiru.application.api.main.MainView;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;

/**
 * Presenter factory for common application functionality.
 */
public interface ApplicationPresenterFactory {
        /**
         * Builds the presenter for main window.
         *
         * @return The presenter
         */
        Presenter<MainView> buildMainPresenter();

        /**
         * Builds the presenter for about window.
         *
         * @return The presenter
         */
        Presenter<AboutView> buildAboutPresenter();
}
```

and for application views:

```
package ar.com.oxen.nibiru.application.api;

import ar.com.oxen.nibiru.application.api.about.AboutView;
import ar.com.oxen.nibiru.application.api.main.MainView;

/**
 * View factory for common application functionality.
 */
public interface ApplicationViewFactory {
        /**
         * Builds the view for main window.
         *
         * @return The view
         */
        MainView buildMainView();

        /**
         * Builds the view for about window.
         *
         * @return The view
```

```
         */
        AboutView buildAboutView();
}
```

7.1 Generic implementation

The ar.com.oxen.nibiru.application.generic bundle provides a generic implementation of basic application components.

The ar.com.oxen.nibiru.application.generic.presenter and ar.com.oxen.nibiru.application.generic.view bundles provide, respectively, generic implementations for application presenters and views.

8 Extension points

Interfaces for extension points are found in the ar.com.oxen.nibiru.extensionpoint.api bundle. The design is simple: each extension point has just an interface and a name. Besides, the extensions can be enabled or disabled at runtime.

To perform an action whenever an extension is added or removed, the ExtensionTracker interface must be used :

```
package ar.com.oxen.nibiru.extensionpoint.api;

/**
 * Callback for tracking extension status.
 *
 * @param <T>
 *            The extension type
 */
public interface ExtensionTracker<T> {
        /**
         * Callback method called when a new extension is registered.
         *
         * @param extension
         *            The extension
         */
        void onRegister(T extension);

        /**
         * Callback method called when an existing extension is unregistered.
         *
         * @param extension
         *            The extension
```

```
         */
        void onUnregister(T extension);
}
```

which provides the necessary callbacks for those events. The ExtensionTrackers must be registered with the ExtensionPointManager service:

```
package ar.com.oxen.nibiru.extensionpoint.api;

/**
 * Service for managing extensions.
 */
public interface ExtensionPointManager {
        /**
         * Registers an extension under a name and an interface
         *
         * @param <K>
         *              The extension point interface
         * @param extension
         *              The extension
         * @param extensionPointName
         *              The extension point name
         * @param extensionPointInterface
         *              The extension point interface
         */
        <K> void registerExtension(K extension, String extensionPointName,
                        Class<K> extensionPointInterface);

        /**
         * Un-registers an extension.
         *
         * @param extension
         *              The extension.
         */
        void unregisterExtension(Object extension);

        /**
         * Registers a tracker for a given extension type and name.
         *
         * @param <T>
         *              The type parametrized on the tracker
         * @param <K>
         *              The extension point interface
         * @param tracker
         *              The tracker
         * @param extensionPointName
         *              The extension point name
```

```
     * @param extensionPointInterface
     *              The extension point interface
     */
    <T, K extends T> void registerTracker(ExtensionTracker<T> tracker,
                        String extensionPointName, Class<K> extensionPointInterf
}
```

The ExtensionPointManager also provides methods for registering new extensions and unregistering a existing one.

8.1 Generic implementation

The ar.com.oxen.nibiru.extensionpoint.generic bundle provides a generic extension point implementation which can be used on both, OSGi and non-OSGi environments.

9 Event bus

Several modules use an event bus. The event bus is accessed using the ar.com.oxen.commons.eventbus.api.Event
interface, which does not belong to Nibiru project but to Oxen Java Commons.
In this project there is also a (pretty) simple implementation of such interface.

10 Modules

As mentioned earlier, the framework is designed so that the functionality can be added as separate modules.

The ar.com.oxen.nibiru.module.utils project provides utility classes for this purpose. Typically, each module will have a component responsible for configuring this module at startup. To that end, this project provides the AbstractModuleConfigurator class, which can be extended in order to create such configurators.

```
package ar.com.oxen.nibiru.module.utils;

import java.util.Collection;
import java.util.LinkedList;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointManager;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;
import ar.com.oxen.nibiru.ui.api.mvp.View;

/**
```

```
 * Base class for module configurators.
 *
 * @param <VF>
 *             The view factory class
 * @param <PF>
 *             The presenter factory class
 */
public abstract class AbstractModuleConfigurator<VF, PF> {
        private ExtensionPointManager extensionPointManager;
        private Collection<Object> registeredExtensions = new LinkedList<Object>
        private EventBus eventBus;
        private VF viewFactory;
        private PF presenterFactory;

        /**
         * Starts the module. This method must be externally called (for example
         * with init-method attribute on Spring context XML).
         */
        public void startup() {
                this.eventBus.subscribeAnnotatedObject(this);
                this.configure();
        }

        /**
         * Same as startup, but for shutdown.
         */
        public void shutdown() {
                /* Custom configuration shutdown for subclasses */
                this.unconfigure();

                this.eventBus.unsubscribeAnnotatedObject(this);

                /* Remove all the extensions */
                for (Object extension : this.registeredExtensions) {
                        this.extensionPointManager.unregisterExtension(extension
                }
                this.registeredExtensions.clear();
        }

        /**
         * Abstract method to be override in order to customize module
         * configuration.
         */
        protected void configure() {
        }
```

```
/**
 * Abstract method to be override in order to customize module
 * un-configuration.
 */
protected void unconfigure() {
}

/**
 * Activates a view/presenter. Typically this method will be called from
 * subclasses upon the receiving of an event from the bus in order to
 * navigate to a given window.
 *
 * @param <V>
 *            The view type
 * @param view
 *            The view
 * @param presenter
 *            The presenter
 */
protected <V extends View> void activate(V view, Presenter<V> presenter)
        presenter.setView(view);
        presenter.go();
        view.show();
}

/**
 * Registers an extension under a name and an interface. The extension u
 * be automatically un-published when the module will be unloaded.
 *
 * @param <K>
 *            The extension point interface
 * @param extension
 *            The extension
 * @param extensionPointName
 *            The extension point name
 * @param extensionPointInterface
 *            The extension point interface
 */
protected <K> void registerExtension(K extension,
                  String extensionPointName, Class<K> extensionPointInterf
        this.extensionPointManager.registerExtension(extension,
                        extensionPointName, extensionPointInterface);
        this.registeredExtensions.add(extension);
}

public void setEventBus(EventBus eventBus) {
```

```
            this.eventBus = eventBus;
        }

        protected EventBus getEventBus() {
            return eventBus;
        }

        protected VF getViewFactory() {
            return viewFactory;
        }

        public void setViewFactory(VF viewFactory) {
            this.viewFactory = viewFactory;
        }

        protected PF getPresenterFactory() {
            return presenterFactory;
        }

        public void setPresenterFactory(PF presenterFactory) {
            this.presenterFactory = presenterFactory;
        }

        protected ExtensionPointManager getExtensionPointManager() {
            return extensionPointManager;
        }

        public void setExtensionPointManager(
                    ExtensionPointManager extensionPointManager) {
            this.extensionPointManager = extensionPointManager;
        }
}
```

You should inject all the required dependencies and trigger the startup() method on startup. On shutdown, you should trigger the shutdown() method. In order to provide custom startup/shutdown configuration logic, you can override the configure() and unconfigure() methods.

Typically, this component will set up navigation between different module screens. For this end, the AbstractModuleConfigurator class provides access to the event bus (which must be injected) and sets itself as listener on that bus. So you can add event handling methods annotated with @EventHandler. In order to show a given view/presenter, you can use the activate() method.

Also, the class provides methods for registering extension points (the ExtensionPointManager must be injected). This is helpful, since the extensions are automatically unregistered when the module is down.

Regarding menus, they are implemented via extension points. So it is only necessary to register an extension with the following interface:

```
package ar.com.oxen.nibiru.ui.api.extension;

/**
 * Extension that represents an item on the menu.
 */
public interface MenuItemExtension {
        /**
         * @return The item name
         */
        String getName();

        /**
         * @return The position (lower numbers are shown first)
         */
        int getPosition();

        /**
         * Method to be executed when the menu is created.
         */
        void onClick();

        /**
         * @return Roles which this extension is available
         */
         String[] getAllowedRoles();
}
```

or with the following one:

```
package ar.com.oxen.nibiru.ui.api.extension;

/**
 * Extension that represents a menu that can contain other menus.
 */
public interface SubMenuExtension {
        /**
         * @return The sub-menu name
         */
        String getName();

        /**
         * @return The position (lower numbers are shown first)
         */
        int getPosition();
```

```
        /**
         * @return The extension point name where entries of this sub-menu shoul
         *         added.
         */
        String getExtensionPoint();

        /**
         * @return Roles which this extension is available
         */
         String[] getAllowedRoles();

}
```

You must define an extension point name for each menu. The extension point to the main menu is ar.com.oxen.nibiru.menu.

The getAllowedRoles method indicates the required roles in order to execute the menu. Such roles are validated against security services. If no role is specified (or null is returned), no validation is done (so everybody can execute the menu).

It is worth noting that the ar.com.oxen.nibiru.ui.utils bundle contains simple implementations of these interfaces.

11 Session

Applications usually have some kind of session information. This is, data that are specific to the user that is connected at any given time. Typically, in a Web application, this information is stored in the HTTP session.

To support the goal of keeping the various components decoupled from the implementation, the ar.com.oxen.nibiru.session.api project provides a generic interface for the session.

```
package ar.com.oxen.nibiru.session.api;

/**
 * Component holding session data.
 */
public interface Session {
        /**
         * Gets an object from session data.
         *
         * @param <T>
         *            The object type
         * @param key
```

```
     *            The object key (must be unique)
     * @return The object
     */
    <T> T get(String key);

    /**
     * Puts an object into session data.
     *
     * @param key
     *            The object key (must be unique)
     * @param value
     *            The object
     */
    void put(String key, Object value);

    /**
     * Removes an object from session data.
     *
     * @param key
     *            The object key (must be unique)
     */
    void remove(String key);

    /**
     * @return An String identifying the session.
     */
    String getId();

    /**
     * @return A mutex that can be used in order to synchronize concurrent
     *         (threaded) session access
     */
    Object getMutex();

    /**
     * Registers a listener for session destruction.
     *
     * @param name
     *            The callback name (must be unique)
     * @param callback
     *            The callback
     */
    void registerDestructionCallback(String name, Runnable callback);

    /**
     * @return True if the session is valid
```

```
         */
        boolean isValid();
}
```

11.1 HTTP implementation

The ar.com.oxen.nibiru.session.http project provides access to the HTTP session using components from ar.com.oxen.nibiru.http.utils. In the webapp, you must place a filter of type ar.com.oxen.nibiru.http.utils.SessionHolderFilter in order to make HTTP session accessible via ar.com.oxen.nibiru.http.utils.SessionHolder. For more details, look at the sample app.

12 Conversations

A common scenario in business applications includes users operating on a set of data for a given time interval and finally confirming or cancelling pending operations. The conversation (ar.com.oxen.nibiru.conversation.api project) serves as an abstraction of this concept:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Interface representing a conversation between the user and the application.
 */
public interface Conversation {
        /**
         * Finishes the conversation OK. Typically, this action is called when a
         * user clicks an "accept" button in order to confirm database changes,
         */
        void end();

        /**
         * Cancels the conversation. Typically called when the user presses a
         * "cancel" button.
         */
        void cancel();

        /**
         * Registers a conversation status tracker.
         *
         * @param tracker
         *            The tracker
         */
```

```
    void registerTracker(ConversationTracker tracker);

    /**
     * Activates the conversation and executes the code provided by the
     * callback. Code called from the callback can access the conversation u
     * the {@link ConversationAccessor} service.
     *
     * @param <T>
     *            The type to be returned by the callback
     * @param callback
     *            The callback
     * @return The object returned by the callback
     */
    <T> T execute(ConversationCallback<T> callback);

    /**
     * Gets an object from conversation data.
     *
     * @param <T>
     *            The object type
     * @param key
     *            The object key (must be unique)
     * @return The object
     */
    <T> T get(String key);

    /**
     * Puts an object into conversation data.
     *
     * @param key
     *            The object key (must be unique)
     * @param value
     *            The object
     */
    void put(String key, Object value);

    /**
     * Removes an object from conversation data.
     *
     * @param key
     *            The object key (must be unique)
     */
    void remove(String key);
}
```

The conversation provides a way to decouple the user interface from the implementation of the various services that require conversation information. For example, suppose you are using the CRUD module with the JPA service implementation. The user interface layer creates a conversation when opening the presenter. With each service call, the CRUD service implementation extracts the active EntityManager from the conversation. Thus, the upper layers doesn't needs to know the details about conversation information needs at lower layers.

To implement this process, the client (usually the presentation layer) creates a conversation using the factory:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Conversation factory.
 */
public interface ConversationFactory {
        /**
         * Builds a new conversation.
         *
         * @return The conversation
         */
        Conversation buildConversation();
}
```

and each time you access a service that requires information from conversation, does it using the execute() method, which receives a callback with a doInConversation() method, which will runs after enabling the conversation:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Conversation callback. Used to run code that can access the active
 * conversation using {@link ConversationAccessor}.
 *
 * @param <T>
 */
public interface ConversationCallback<T> {
        /**
         * Method to be executed when conversation is activated.
         *
         * @param conversation
         *            The active conversation
         * @return Anything that the callback would want to return
         * @throws Exception
         *             At any error
         */
        T doInConversation(Conversation conversation) throws Exception;
```

```
}
```

Finally, the client can invoke the end() or cancel() methods, in order to either finishing or canceling the conversation.

From lower layers, you can access the active conversation through ConversationAccessor service:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Service used to access current active conversation.
 */
public interface ConversationAccessor {
        /**
         * @return The current conversation
         */
        Conversation getCurrentConversation();
}
```

Using get() and put() methods, the component can read and write values from/into the conversation. If you want to perform an action when the conversation terminates/cancels, you can use the registerTracker() to register a callback:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Listener for tracking conversation life cycle.
 *
 */
public interface ConversationTracker {
        /**
         * Called when conversation finishes OK.
         *
         * @param conversation
         *              The finished conversation
         */
        void onEnd(Conversation conversation);

        /**
         * Called when conversation is canceled.
         *
         * @param conversation
         *              The canceled conversation
         */
        void onCancel(Conversation conversation);
}
```

The idea of establishing a mechanism comes from Seam conversations, but some modifications were made. First, we aimed to make a simpler design and not being oriented specifically to Web applications. For example, Seam conversations are hierarchical, while those of Nibiru are not. We even hade the idea of unifying the concept of conversation with the session and make it hierachical (being the session the main conversation), but this would add complexity to conversation semantics and force an awkward interface unification, without providing benefits.

12.1 Generic impementation

The ar.com.oxen.nibiru.conversation.generic module contains generic conversation services implementations.

```
package ar.com.oxen.nibiru.conversation.generic;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import ar.com.oxen.nibiru.conversation.api.Conversation;
import ar.com.oxen.nibiru.conversation.api.ConversationCallback;
import ar.com.oxen.nibiru.conversation.api.ConversationTracker;

public class GenericConversation implements Conversation {
        private Set<ConversationTracker> trackers = new HashSet<ConversationTrac
        private Map<String, Object> attributes = new HashMap<String, Object>();
        private GenericConversationManager conversationManager;

        public GenericConversation(GenericConversationManager conversationManage
                super();
                this.conversationManager = conversationManager;
        }

        @Override
        public void end() {
                for (ConversationTracker tracker : this.trackers) {
                        tracker.onEnd(this);
                }

                this.trackers.clear();
                this.attributes.clear();
        }

        @Override
```

```
        public void cancel() {
                for (ConversationTracker tracker : this.trackers) {
                        tracker.onCancel(this);
                }

                this.trackers.clear();
                this.attributes.clear();
        }

        @Override
        public void registerTracker(ConversationTracker tracker) {
                this.trackers.add(tracker);
        }

        @Override
        public <T> T execute(ConversationCallback<T> callback) {
                try {
                        Conversation previousConversation = this.conversationMan
                                        .getCurrentConversation();
                        this.conversationManager.setCurrentConversation(this);
                        T returnValue = callback.doInConversation(this);
                        this.conversationManager
                                        .setCurrentConversation(previousConversa
                        return returnValue;
                } catch (Exception e) {
                        throw new RuntimeException(e);
                }
        }

        @SuppressWarnings("unchecked")
        @Override
        public <T> T get(String key) {
                return (T) this.attributes.get(key);
        }

        @Override
        public void put(String key, Object value) {
                this.attributes.put(key, value);
        }

        @Override
        public void remove(String key) {
                this.attributes.remove(key);
        }
}
```

```
package ar.com.oxen.nibiru.conversation.generic;

import ar.com.oxen.nibiru.conversation.api.Conversation;
import ar.com.oxen.nibiru.conversation.api.ConversationAccessor;
import ar.com.oxen.nibiru.conversation.api.ConversationFactory;

public class GenericConversationManager implements ConversationFactory, Conversa
        private ThreadLocal<Conversation> currentConversation = new ThreadLocal<

        @Override
        public Conversation buildConversation() {
                return new GenericConversation(this);
        }

        @Override
        public Conversation getCurrentConversation() {
                return this.currentConversation.get();
        }

        void setCurrentConversation(Conversation conversation) {
                this.currentConversation.set(conversation);
        }

}
```

13 Persistence

13.1 JPA

JPAis used for persistence. While there are mutliple persistence mechanisms in Java, JPA is the most widespread. For this reason, this specification was chosen over other mechanisms. However, nothing prevents from implementing persistence services using a different technology (of course, this would imply implementing again the modules which depend on JPA).

Since JPA is an API itself, no Nibiru-specific API was defined for object persistence. On the other hand, instances of javax.persistence.EntityManagerFactory, from JPA specification, are exposed as services.

On Karaf container, when a bundle with JPA configuration is deployed, an EntityManagerFactory is automatically created and exposed as mentioned before. Aries is responsible for doing this task. For an explanation about which configuration and files must be included into de bundle, read the Aries JPA documentation.

In some scenarios (such as CRUD), a conversation bound EntityManager would be useful. This way, you can hold changes until the conversation is finished. Also, you can avoid dealing with disconnected objects.

For this purpose, the ar.com.oxen.nibiru.jpa bundle provides the ConversationEntityManagerFactory class, which wraps an EntityManagerFactory and bounds each EntityManager created to the conversation. A proxy that looks for the EntityManager in the conversation is returned, so you can inject it directly on your component and use it without caring about conversation.

As said before, Aries JPA exposes an EntityManagerFactory for each JPA bundle. In order to integrate this approach with the conversation classes, the ar.com.oxen.nibiru.jpa.blueprint bundle implements an extender (ConversationEntityManagerExtender) that listens when an EntityManagerFactory service is created, wraps it into a ConversationEntityManagerFactory and exposes it as a new service. The osgi.unit.name property (which contains the persistence unit name) is suffixed with "_conversation" in order to diferentiate it from the wrapped service.

13.2 Database

Regarding database access, a javax.sql.DataSource service is exposed . In this case it was not necessary to define a specific Nibiru API. The ar.com.oxen.nibiru.datasource.dbcp bundle provides an implementation using DBCP. On the other hand, the ar.com.oxen.nibiru.datasource.c3p0 bundle provides a c3p0 implementation.

The database connection settings and JDBC driver visibility are also added as OSGi fragments. Look at ar.com.oxen.nibiru.sample.datasource.fragment project for an example. c3p0 implementatio requires an extra fragment, in order to make JDBC driver class visible to c3p0 bundle.

14 User interface

The ar.com.oxen.nibiru.ui.api bundle contains interfaces for presentation layer. The approach aims to build the view using the MVP pattern (passive view). Within the package we have 3 main sub-packages:

1. extension: Contains interfaces to be implemented by UI extensions (currently sub-menu and menu - see Modules section for details).
2. mvp: Contains the interfaces used to implement the MVP pattern: Presenter, View and all necessary ones in order to access to data and events (HasValue, HasClickHandler, clickHandler, etc.). .
3. view: Contains interfaces for view component abstraction. These interfaces are used every time you want to access to a specific widget in a

generic way. For example, a button or text field. The idea is to have adapters for the widgets of different UI technologies.

Using this approach, the user has two options for creating a view:

1. In a generic way, ie using an implementation of ar.com.oxen.nibiru.ui.api.view.ViewFactory in order to access generic widget interfaces. This way, a limited user interface can be built, but you can easily change the subjacent technology.
2. Using a specific technology and making the view class implementing the interface used in the MVP. This way you can take advantage of technology characteristics and use graphic editors. In contrast, the changing the technology mean more work.

As the proposed MVP model is passive view, the presenter simply has a reference to an interface that represents the view (at Google the term Display is used). This lets you use either one of the two approaches, without changing the presenter.

In summary, the main MVP interfaces are Presenter:

```
package ar.com.oxen.nibiru.ui.api.mvp;

/**
 * A presenter. The presenter should contain the presentation logic, in order to
 * keep it decoupled from the view.
 *
 * @param <V>
 *            The view type.
 */
public interface Presenter<V extends View> {
        /**
         * Activates the presenter. This method is called after setting the view
         * Typically, this method will add listeners for presentation logic that
         * reacts to view events.
         */
        void go();

        /**
         * @param view
         *            The view to be used with the presenter
         */
        void setView(V view);
}
```

and View:

```
package ar.com.oxen.nibiru.ui.api.mvp;

/**
 * A view. Implementations of this interface shouldn't contain presentation
 * logic. Instead, display-related logic, such as layout setup, text
 * internationalization, etc should be responsibility of View implementations.
 *
 */
public interface View {
        /**
         * Shows the view.
         */
        void show();

        /**
         * Closes the view.
         */
        void close();
}
```

The presentation logic should be put on the method go() of Presenter class.

Widgets abstraction interfaces (ar.com.oxen.nibiru.ui.api.view package) are varied. But all should be instantiated by an implementation of ViewFactory:

```
package ar.com.oxen.nibiru.ui.api.view;

/**
 * Builds components (widgets, windows, etc) to be used in views. The purpose of
 * this interface is hiding UI framework specific implementations.
 */
public interface ViewFactory {
        /**
         * Builds a main window.
         *
         * @return The main window.
         */
        MainWindow buildMainWindow();

        /**
         * Builds a window.
         *
         * @return The window
         */
        Window buildWindow();

        /**
```

```
     * Builds a label.
     *
     * @param <T>
     *            The type of data to be shown by the label. Typically Strin
     * @param type
     *            The class of data to be shown by the label. Typically Stri
     * @return The label
     */
    <T> Label<T> buildLabel(Class<T> type);

    /**
     * Builds a button.
     *
     * @return The button.
     */
    Button buildButton();

    /**
     * Builds a text field.
     *
     * @param <T>
     *            The type of data to be shown by the text field. Typically
     *            String.
     * @param type
     *            The class of data to be shown by the text field. Typically
     *            String.
     * @return The text field
     */
    <T> TextField<T> buildTextField(Class<T> type);

    /**
     * Builds a password field.
     *
     * @param <T>
     *            The type of data to be shown by the password field. Typica
     *            String.
     * @param type
     *            The class of data to be shown by the password field. Typic
     *            String.
     * @return The password field
     */
    <T> PasswordField<T> buildPasswordField(Class<T> type);

    /**
     * Builds a multiline text area.
     *
```

```
     * @param <T>
     *             The type of data to be shown by the password field. Typica
     *             String.
     * @param type
     *             The class of data to be shown by the password field. Typic
     *             String.
     * @return The text area
     */
    <T> TextArea<T> buildTextArea(Class<T> type);

    /**
     * Builds a date field.
     *
     * @return The date field
     */
    DateField buildDateField();

    /**
     * Builds a time field.
     *
     * @return The time field
     */
    TimeField buildTimeField();

    /**
     * Builds a check box.
     *
     * @return The check box
     */
    CheckBox buildCheckBox();

    /**
     * Builds a combo box.
     *
     * @param <T>
     *             The type of data to be shown by the combo.
     * @param type
     *             The class of data to be shown by the combo.
     * @return The combo box
     */
    <T> ComboBox<T> buildComboBox(Class<T> type);

    /**
     * Builds a list select.
     *
     * @param <T>
```

```
     *            The type of data to be shown by the list select.
     * @param type
     *            The class of data to be shown by the list select.
     * @return The list select
     */
    <T> ListSelect<T> buildListSelect(Class<T> type);

    /**
     * Builds a table.
     *
     * @return The table
     */
    Table buildTable();

    /**
     * Builds a panel with vertical layout.
     *
     * @return The panel.
     */
    Panel buildVerticalPanel();

    /**
     * Builds a panel with horizontal layout.
     *
     * @return The panel.
     */
    Panel buildHorizontalPanel();

    /**
     * Builds a panel with form layout.
     *
     * @return The panel.
     */
    FormPanel buildFormPanel();

    /**
     * Builds a tabbed panel.
     *
     * @return The panel
     */
    Panel buildTabPanel();

    /**
     * Builds an embedded.
     *
     * @return The embedded
```

```
         */
        Embedded buildEmbedded();

        /**
         * Builds a contextual menu.
         *
         * @return The context menu.
         */
        ContextMenu buildContextMenu();
}
```

14.1 Vaadin implementation

The ar.com.oxen.nibiru.ui.vaadin project contains a factory and its associated adapters required in order to implement ar.com.oxen.nibiru.ui.api.view interfaces using Vaadin.

It also provides an specific Vaadin application for Nibiru:

```
package ar.com.oxen.nibiru.ui.vaadin.application;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.commons.eventbus.api.EventHandler;
import ar.com.oxen.nibiru.application.api.ApplicationStartEvent;
import ar.com.oxen.nibiru.application.api.ApplicationThemeChangeEvent;
import ar.com.oxen.nibiru.i18n.api.LocaleHolder;

import com.vaadin.Application;

public class NibiruApplication extends Application {
        /**
         * Serial ID.
         */
        private static final long serialVersionUID = -8241304827319878154L;
        private EventBus eventBus;
        private LocaleHolder localeHolder;
        private EventHandler<ApplicationThemeChangeEvent> applicationThemeChange

        @Override
        public void init() {
                this.localeHolder.setLocale(this.getLocale());

                this.applicationThemeChangeEventHandler = new EventHandler<Appli
                        @Override
                        public void onEvent(ApplicationThemeChangeEvent event) {
                                setTheme(event.getTheme());
```

```
                        }
                };
                this.eventBus.addHandler(ApplicationThemeChangeEvent.class,
                                this.applicationThemeChangeEventHandler);

                this.eventBus.fireEvent(new ApplicationStartEvent());
        }

        @Override
        public void close() {
                this.eventBus.removeHandler(this.applicationThemeChangeEventHand
        }

        public void setEventBus(EventBus eventBus) {
                this.eventBus = eventBus;
        }

        public void setLocaleHolder(LocaleHolder localeHolder) {
                this.localeHolder = localeHolder;
        }

}
```

As you can see, if you want to change the Vaadin theme, you can do it by firing an ApplicationThemeChangeEvent on the event bus.

Since Vaadin application can't be exposed as an OSGi service (OSGi services are exposed using Java interfaces, while Vaadin application is a concrete class), Nibiru provides an interface for accessing such component:

```
package ar.com.oxen.nibiru.ui.vaadin.api;

import com.vaadin.Application;

/**
 * Interface for accessing Vaadin application from a service. Since
 * {@link Application} is not an interface, it can't be exposed as a service.
 */
public interface ApplicationAccessor {
        Application createApplication();

        Application getApplication();
}
```

And, as expected, a simple implementation:

```
package ar.com.oxen.nibiru.ui.vaadin.application;
```

```
import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.http.utils.SessionHolder;
import ar.com.oxen.nibiru.i18n.api.LocaleHolder;
import ar.com.oxen.nibiru.ui.vaadin.api.ApplicationAccessor;

import com.vaadin.Application;
import com.vaadin.terminal.gwt.server.WebApplicationContext;

public class SimpleApplicationAccessor implements ApplicationAccessor {
        private EventBus eventBus;
        private LocaleHolder localeHolder;

        @Override
        public Application createApplication() {
                NibiruApplication nibiruApplication = new NibiruApplication();
                nibiruApplication.setEventBus(this.eventBus);
                nibiruApplication.setLocaleHolder(this.localeHolder);
                return nibiruApplication;
        }

        @Override
        public Application getApplication() {
                WebApplicationContext context = WebApplicationContext
                                .getApplicationContext(SessionHolder.getSession(

                if (context.getApplications().size() > 0) {
                        return context.getApplications().iterator().next();
                } else {
                        throw new IllegalStateException("No Vaadin App on contex
                }
        }

        public void setEventBus(EventBus eventBus) {
                this.eventBus = eventBus;
        }

        public void setLocaleHolder(LocaleHolder localeHolder) {
                this.localeHolder = localeHolder;
        }
}
```

14.2 UI utilities

The ar.com.oxen.nibiru.ui.utils project contains generic classes for use in the user interface. Mostly contains abstract classes to be used as base for presenters

, views, extensions, etc. But also contains decorators and generic use classes.

- ar.com.oxen.nibiru.ui.utils.dialog: Contains classes for handling dialogs.
  - For example, the DialogBuilder class allows creating a custom-made modal window:

```
package ar.com.oxen.nibiru.ui.utils.dialog;

import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;
import ar.com.oxen.nibiru.ui.api.view.Button;
import ar.com.oxen.nibiru.ui.api.view.Label;
import ar.com.oxen.nibiru.ui.api.view.Panel;
import ar.com.oxen.nibiru.ui.api.view.ViewFactory;
import ar.com.oxen.nibiru.ui.api.view.Window;

public class DialogBuilder {
        private ViewFactory viewFactory;
        private Window window;
        private Panel messagePanel;;
        private Panel buttonPanel;;

        public DialogBuilder(ViewFactory viewFactory) {
                super();
                this.viewFactory = viewFactory;

                this.window = viewFactory.buildWindow();
                this.window.setModal(true);

                this.messagePanel = viewFactory.buildVerticalPanel();
                this.window.addComponent(this.messagePanel);

                this.buttonPanel = viewFactory.buildHorizontalPanel();
                this.window.addComponent(this.buttonPanel);
        }

        public DialogBuilder title(String title) {
                this.window.setValue(title);
                return this;
        }

        public DialogBuilder message(String message) {
                Label<String> label = this.viewFactory.buildLabel(String.cla
                label.setValue(message);
                this.messagePanel.addComponent(label);
                return this;
```

```
        }

        public DialogBuilder button(String caption) {
                return this.button(caption, null);
        }

        public DialogBuilder button(String caption, final ClickHandler handl
                Button button = this.viewFactory.buildButton();
                button.setValue(caption);
                button.setClickHandler(new ClickHandler() {
                        @Override
                        public void onClick() {
                                window.close();
                                if (handler != null) {
                                        handler.onClick();
                                }
                        }
                });
                this.buttonPanel.addComponent(button);
                return this;
        }

        public Window build() {
                return this.window;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.extension: Provides common UI extension implementations.

  - SimpleMenuItemExtension is an implementation for menu items:

```
package ar.com.oxen.nibiru.ui.utils.extension;

import ar.com.oxen.nibiru.ui.api.extension.MenuItemExtension;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;

public class SimpleMenuItemExtension implements MenuItemExtension {
        private String name;
        private int position;
        private ClickHandler clickHandler;
        private String[] allowedRoles;

        public SimpleMenuItemExtension() {
                super();
        }
```

```
public SimpleMenuItemExtension(String name, int position,
                ClickHandler clickHandler) {
        this(name, position, clickHandler, null);
}

public SimpleMenuItemExtension(String name, int position,
                ClickHandler clickHandler, String[] allowedRoles) {
        super();
        this.name = name;
        this.position = position;
        this.clickHandler = clickHandler;
        this.allowedRoles = allowedRoles;
}

@Override
public String getName() {
        return this.name;
}

@Override
public void onClick() {
        this.clickHandler.onClick();
}

public void setName(String name) {
        this.name = name;
}

public void setClickHandler(ClickHandler clickHandler) {
        this.clickHandler = clickHandler;
}

@Override
public int getPosition() {
        return position;
}

public void setPosition(int position) {
        this.position = position;
}

@Override
public String[] getAllowedRoles() {
        return this.allowedRoles;
}
```

```
}
```

- SimpleSubMenuExtension, in a similar way, implements a sub-menu extension:

```
package ar.com.oxen.nibiru.ui.utils.extension;

import ar.com.oxen.nibiru.ui.api.extension.SubMenuExtension;

public class SimpleSubMenuExtension implements SubMenuExtension {
        private String name;
        private String extensionPoint;
        private int position;
        private String[] allowedRoles;

        public SimpleSubMenuExtension() {
                super();
        }
        public SimpleSubMenuExtension(String name, String extensionPoint,
                        int position) {
                this(name, extensionPoint, position, null);
        }
        public SimpleSubMenuExtension(String name, String extensionPoint,
                        int position, String[] allowedRoles) {
                super();
                this.name = name;
                this.extensionPoint = extensionPoint;
                this.position = position;
                this.allowedRoles = allowedRoles;
        }

        public void setName(String name) {
                this.name = name;
        }

        public void setExtensionPoint(String extensionPoint) {
                this.extensionPoint = extensionPoint;
        }

        @Override
        public String getName() {
                return this.name;
        }

        @Override
        public String getExtensionPoint() {
```

```
            return this.extensionPoint;
        }

        @Override
        public int getPosition() {
            return position;
        }

        public void setPosition(int position) {
            this.position = position;
        }
        @Override
        public String[] getAllowedRoles() {
            return this.allowedRoles;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.mvp: Contains utility classes for implementing the MVP pattern.
  - AbstractEventBusClickHandler is a base class for click handlers which fires events on the bus:

```
package ar.com.oxen.nibiru.ui.utils.mvp;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;

public abstract class AbstractEventBusClickHandler implements ClickHandler {
        private EventBus eventBus;

        public AbstractEventBusClickHandler() {
            super();
        }

        public AbstractEventBusClickHandler(EventBus eventBus) {
            super();
            this.eventBus = eventBus;
        }

        public void setEventBus(EventBus eventBus) {
            this.eventBus = eventBus;
        }

        protected EventBus getEventBus() {
            return eventBus;
```

```
        }

}
```

- AbstractPresenter is a base class for any presenter:

```
package ar.com.oxen.nibiru.ui.utils.mvp;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;
import ar.com.oxen.nibiru.ui.api.mvp.HasCloseWidget;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;
import ar.com.oxen.nibiru.ui.api.mvp.View;

public abstract class AbstractPresenter<V extends View> implements Presenter
        private V view;
        private EventBus eventBus;

        protected AbstractPresenter(EventBus eventBus) {
                super();
                this.eventBus = eventBus;
        }

        @Override
        public void setView(V view) {
                this.view = view;
        }

        protected V getView() {
                return view;
        }

        protected EventBus getEventBus() {
                return eventBus;
        }

        protected void configureClose(HasCloseWidget hasCloseWidget) {
                hasCloseWidget.getCloseHandler().setClickHandler(new ClickHa
                        @Override
                        public void onClick() {
                                getView().close();
                        }
                });
        }
}
```

- HasValueI18nDecorator is a wrapper for HasValue<String> instances, which performs translation on the text using i18n services:

```
package ar.com.oxen.nibiru.ui.utils.mvp;

import ar.com.oxen.nibiru.i18n.api.MessageSource;
import ar.com.oxen.nibiru.ui.api.mvp.HasValue;

public class HasValueI18nDecorator implements HasValue<String> {
        private HasValue<String> decorated;
        private MessageSource messageSource;
        private String code;

        public HasValueI18nDecorator(HasValue<String> decorated,
                        MessageSource messageSource) {
                super();
                this.decorated = decorated;
                this.messageSource = messageSource;
        }

        @Override
        public String getValue() {
                return this.code;
        }

        @Override
        public void setValue(String value) {
                this.code = value;
                this.decorated.setValue(this.messageSource.getMessage(this.c
        }

}
```

- SimpleEventBusClickHandler is an event handler that fires an event on the bus with the specified class and topic:

```
package ar.com.oxen.nibiru.ui.utils.mvp;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.commons.exception.api.ExceptionWrapper;

public class SimpleEventBusClickHandler extends AbstractEventBusClickHandler
        private Class<?> eventClass;
        private String topic;

        public SimpleEventBusClickHandler() {
```

```
                super();
        }

        public SimpleEventBusClickHandler(EventBus eventBus, Class<?> eventC
                        String topic) {
                super(eventBus);
                this.eventClass = eventClass;
                this.topic = topic;
        }

        @Override
        public void onClick() {
                try {
                        this.getEventBus().fireEvent(this.eventClass.newInst
                                        this.topic);
                } catch (InstantiationException e) {
                        throw new ExceptionWrapper(e);
                } catch (IllegalAccessException e) {
                        throw new ExceptionWrapper(e);
                }
        }

        public void setEventClass(Class<?> eventClass) {
                this.eventClass = eventClass;
        }

        public void setTopic(String topic) {
                this.topic = topic;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.view: Provides base classes for defining views.
  - AbstractAdapter represents a generic view adapter:

```
package ar.com.oxen.nibiru.ui.utils.view;

public class AbstractAdapter<T> {
        private T adapted;

        public AbstractAdapter(T adapted) {
                super();
                this.adapted = adapted;
        }

        public T getAdapted() {
```

```
                return adapted;
        }
}
```

- AbstractWindowViewAdapter is a base class for window-based views:

```
package ar.com.oxen.nibiru.ui.utils.view;

public class AbstractAdapter<T> {
        private T adapted;

        public AbstractAdapter(T adapted) {
                super();
                this.adapted = adapted;
        }

        public T getAdapted() {
                return adapted;
        }
}
```

15 Security

15.1 Security API

The interfaces required for accessing security services (authentication and authorization) are found in the ar.com.oxen.nibiru.security.api project . Currently user/password authentication and key role authorization are supported.

Authentication is done through the AuthenticationService interface:

```
package ar.com.oxen.nibiru.security.api;

/**
 * Service for authenticating users.
 */
public interface AuthenticationService {
        /**
         * Performs an user log-on.
         *
         * @param username
         *              The user name
         * @param password
         *              The password
```

```
         * @throws BadCredentialsException
         *             If the user name and/or the password is not valid
         */
        void login(String username, String password) throws BadCredentialsExcept

        /**
         * Performs an user log-off.
         */
        void logout();

        /**
         * @return The login name of the logged user (if any).
         */
        String getLoggedUserName();
}
```

While authorization is performed by AuthorizationService:

```
package ar.com.oxen.nibiru.security.api;

/**
 * Service for authorizing actions and users.
 */
public interface AuthorizationService {
        /* Defualt roles */
        String OPERATOR_ROLE = "ar.com.oxen.nibiru.security.role.Operator";
        String ADMINISTRATOR_ROLE = "ar.com.oxen.nibiru.security.role.Administra

        /**
         * Checks if the logged user has a given role.
         *
         * @param role
         *             The role name.
         * @return True if the user has the role
         */
        boolean isCallerInRole(String role);

        /**
         * Checks if an specific user has a given role.
         *
         * @param username
         *             The username
         * @param role
         *             The role name.
         * @return True if the user has the role
         */
        boolean isUserInRole(String username, String role);
```

```
}
```

Profile information, such as first and last name, can be accessed through the following interface:

```
package ar.com.oxen.nibiru.security.api;

public interface Profile {
        boolean isActive();

        String getUsername();

        String getFirstName();

        String getLastName();

        void activate(String username, String firstName, String lastName);

        void deactivate();
}
```

You can simply inject it into your component.

Finally, for hashing purposes (such as storing the user password hash), a hash service is provided:

```
package ar.com.oxen.nibiru.security.api;

public interface HashService {
        String hash(String data);
}
```

15.1.1 Spring Security implementation

The project ar.com.oxen.nibiru.security.spring provides security components implementations using the Spring Security framework.

SpringAuthenticationService performs authentication by delegating on Spring Security's AuthenticationManager:

```
package ar.com.oxen.nibiru.security.spring;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticatio
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
```

```
import ar.com.oxen.nibiru.security.api.AuthenticationService;
import ar.com.oxen.nibiru.security.api.BadCredentialsException;
import ar.com.oxen.nibiru.security.api.Profile;
import ar.com.oxen.nibiru.security.manager.api.SecurityManager;
import ar.com.oxen.nibiru.security.manager.api.UserData;
import ar.com.oxen.nibiru.session.api.Session;

public class SpringAuthenticationService implements AuthenticationService {
        private Session session;
        final static String AUTHENTICATION_KEY = "ar.com.oxen.nibiru.security.sp
        private AuthenticationManager authenticationManager;
        private SecurityManager securityManager;
        private Profile profile;

        @Override
        public void login(String username, String password)
                        throws BadCredentialsException {
                try {
                        Authentication authentication = this.authenticationManag
                                        .authenticate(new UsernamePasswordAuthe
                                                        username, password != nu
                        this.session.put(AUTHENTICATION_KEY, authentication);

                        UserData userData = this.securityManager.getUserData(use
                        this.profile.activate(userData.getUsername(),
                                        userData.getFirstName(), userData.getLast

                } catch (AuthenticationException e) {
                        this.profile.deactivate();
                        throw new BadCredentialsException();
                }
        }

        @Override
        public void logout() {
                this.session.remove(AUTHENTICATION_KEY);
        }

        @Override
        public String getLoggedUserName() {
                UsernamePasswordAuthenticationToken authentication = this.sessio
                                .get(AUTHENTICATION_KEY);
                return authentication.getName();
        }

        public void setSession(Session session) {
```

```
                this.session = session;
        }

        public void setAuthenticationManager(
                        AuthenticationManager authenticationManager) {
                this.authenticationManager = authenticationManager;
        }

        public void setSecurityManager(SecurityManager securityManager) {
                this.securityManager = securityManager;
        }

        public void setProfile(Profile profile) {
                this.profile = profile;
        }
}
```

However, an AuthenticationManager instance should be injected into instances of this class.

Since SpringAuthenticationService stores the authentincation information into the Nibiru session, the SpringAuthorizationService class just reads the authorities from such session:

```
package ar.com.oxen.nibiru.security.spring;

import static ar.com.oxen.nibiru.security.spring.SpringAuthenticationService.AUT

import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;

import ar.com.oxen.nibiru.security.api.AuthorizationService;
import ar.com.oxen.nibiru.security.manager.api.SecurityManager;
import ar.com.oxen.nibiru.security.manager.api.UserData;
import ar.com.oxen.nibiru.session.api.Session;

public class SpringAuthorizationService implements AuthorizationService {
        private SecurityManager securityManager;
        private Session session;

        @Override
        public boolean isCallerInRole(String role) {
                Authentication authentication = this.session.get(AUTHENTICATION_

                if (authentication != null) {
                        for (GrantedAuthority authority : authentication.getAutl
                                if (role.equals(authority.getAuthority())) {
```

```
                                        return true;
                                }
                        }
                }
                return false;
        }

        @Override
        public boolean isUserInRole(String username, String role) {
                // TODO: esto limita los mecanismos de autorizacion y a la vez e
                // usando dos mecanismos distintos en la misma clase!
                return this.hasRole(this.securityManager.getUserData(username),
        }

        private boolean hasRole(UserData userData, String role) {
                if (userData != null) {
                        for (String currentRole : userData.getRoles()) {
                                if (role.equals(currentRole)) {
                                        return true;
                                }
                        }
                }
                return false;
        }

        public void setSession(Session session) {
                this.session = session;
        }

        public void setSecurityManager(SecurityManager securityManager) {
                this.securityManager = securityManager;
        }
}
```

Hashing service is implemented by providing an adapter to Spring Security PasswordEncoder:

```
package ar.com.oxen.nibiru.security.spring;

import org.springframework.security.authentication.encoding.PasswordEncoder;

import ar.com.oxen.nibiru.security.api.HashService;

public class PasswordEncoderHashService implements HashService {
        private PasswordEncoder passwordEncoder;

        @Override
```

```
        public String hash(String data) {
                return this.passwordEncoder.encodePassword(data, null);
        }

        public void setPasswordEncoder(PasswordEncoder passwordEncoder) {
                this.passwordEncoder = passwordEncoder;
        }
}
```

15.2 Security management API

Security management API provides functionality which is not directly related to authentication/authorization issues. Instead, it focuses on data management for supporting these activities. For example, it allows accessing user information.

The main interface for this module is SecurityManager:

```
package ar.com.oxen.nibiru.security.manager.api;

/**
 * Security manager.
 */
public interface SecurityManager {
        /**
         * Changes user's passowrd.
         *
         * @param username
         *              The user name
         * @param oldPassword
         *              The previous password
         * @param newPassword
         *              The new password
         * @throws UserNotFoundException
         *               If no user with the given name is found
         * @throws InvalidOldPassword
         *               If previous password is not valid
         */
        void changePassword(String username, String oldPassword, String newPassw
                        throws UserNotFoundException, InvalidOldPassword;

        /**
         * Retrieves the user data.
         *
         * @param username
         *              The user name
         * @return The user data
```

```
         * @throws UserNotFoundException
         *             If no user with the given name is found
         */
        UserData getUserData(String username) throws UserNotFoundException;
}
```

15.2.1 JPA implementation

The ar.com.oxen.nibiru.security.manager.jpa module provides a JPA implementation for SecurityManager:

```
package ar.com.oxen.nibiru.security.manager.jpa;

import java.util.HashSet;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.persistence.Query;
import javax.persistence.TypedQuery;

import ar.com.oxen.nibiru.security.api.AuthorizationService;
import ar.com.oxen.nibiru.security.manager.api.InvalidOldPassword;
import ar.com.oxen.nibiru.security.manager.api.SecurityManager;
import ar.com.oxen.nibiru.security.manager.api.UserData;
import ar.com.oxen.nibiru.security.manager.api.UserNotFoundException;
import ar.com.oxen.nibiru.security.manager.jpa.domain.Role;
import ar.com.oxen.nibiru.security.manager.jpa.domain.User;

public class JpaSecurityManager implements SecurityManager {
        private EntityManager entityManager;

        @Override
        public void changePassword(String username, String oldPassword,
                        String newPassword) {
                User user = this.findUserByUsername(username);
                String currentPassword = user.getPassword() != null ? user
                                .getPassword() : "";
                String validationPassword = oldPassword != null ? oldPassword :

                if (!currentPassword.equals(validationPassword)) {
                        throw new InvalidOldPassword();
                }

                user.setPassword(newPassword);
```

```
        this.entityManager.persist(user);
}

@Override
public UserData getUserData(String username) throws UserNotFoundExceptio
        this.checkUsers();
        return new UserAdapter(this.findUserByUsername(username));
}

private User findUserByUsername(String username) {
        try {
                Query query = this.entityManager
                                .createQuery("select u from User u where
                query.setParameter("usernameParam", username);
                User user = (User) query.getSingleResult();

                // TODO: check this
                this.entityManager.refresh(user); // Damn Hibernate cach
                return user;
        } catch (NoResultException e) {
                throw new UserNotFoundException();
        }
}

private void checkUsers() {
        TypedQuery<Number> query = this.entityManager.createQuery(
                        "select count(u) from User u", Number.class);

        Number count = query.getSingleResult();

        if (count.intValue() == 0) {
                User admin = new User();
                admin.setFirstName("Admin");
                admin.setLastName("Admin");
                admin.setUsername("admin");
                admin.setPassword("");
                admin.setRoles(new HashSet<Role>());
                admin.getRoles().add(
                                this.findOrCreateRole(AuthorizationServi
                admin.getRoles()
                                .add(this
                                                .findOrCreateRole(Autho
                this.entityManager.persist(admin);
        }
}
```

```
        private Role findOrCreateRole(String roleName) {
                TypedQuery<Role> query = this.entityManager.createQuery(
                                "select r from Role r where r.name = :roleName",

                query.setParameter("roleName", roleName);

                List<Role> roles = query.getResultList();
                Role role;

                if (roles.size() == 0) {
                        role = new Role();
                        role.setName(roleName);
                        role.setDescription(roleName);
                        this.entityManager.persist(role);
                } else {
                        role = roles.get(0);
                }

                return role;
        }

        public void setEntityManager(EntityManager entityManager) {
                this.entityManager = entityManager;
        }
}
```

15.3 Security modules

15.3.1 Default module

The ar.com.oxen.nibiru.security.module bundle provides a security module implementation with:

- A user - role - group domain model.
- Administration over these entities using the CRUD module.

The following configuration file show how the components are wired:

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
                http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/x
```

```
␣␣␣␣␣␣␣␣">

        <service interface="ar.com.oxen.nibiru.i18n.api.MessageProvider"
                ref="securityMessageProvider" />

        <reference id="authenticationService"
                interface="ar.com.oxen.nibiru.security.api.AuthenticationService

        <reference id="authorizationService"
                interface="ar.com.oxen.nibiru.security.api.AuthorizationService"

        <reference id="securityViewFactory"
                interface="ar.com.oxen.nibiru.security.ui.api.SecurityViewFactor

        <reference id="securityPresenterFactory"
                interface="ar.com.oxen.nibiru.security.ui.api.SecurityPresenterF

        <reference id="extensionPointManager"
                interface="ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointM

        <reference id="eventBus" interface="ar.com.oxen.commons.eventbus.api.Eve

        <reference id="crudViewFactory"
                interface="ar.com.oxen.nibiru.crud.ui.api.CrudViewFactory" />

        <reference id="crudPresenterFactory"
                interface="ar.com.oxen.nibiru.crud.ui.api.CrudPresenterFactory"

        <reference id="wrapperFactory" interface="ar.com.oxen.commons.bean.api.W

        <reference id="entityManagerFactory" interface="javax.persistence.EntityM
                filter="(osgi.unit.name=nibiruSecurity_conversation)" />

        <reference id="transactionTemplate"
                interface="ar.com.oxen.nibiru.transaction.api.TransactionTemplat

        <bean id="securityMessageProvider"
                class="ar.com.oxen.nibiru.i18n.generic.ResourceBundleMessageProv
                <property name="baseName"
                        value="ar.com.oxen.nibiru.security.module.ui.i18n.messag
                <property name="resourceClassLoader">
                        <bean class="ar.com.oxen.nibiru.osgi.utils.BundleDelegat
                                <argument ref="blueprintBundle" />
                        </bean>
                </property>
        </bean>
```

```
        <bean init-method="startup" destroy-method="shutdown"
                class="ar.com.oxen.nibiru.security.module.ModuleConfigurator">
                <property name="extensionPointManager" ref="extensionPointManage
                <property name="eventBus" ref="eventBus" />
                <property name="authenticationService" ref="authenticationServic
                <property name="viewFactory" ref="crudViewFactory" />
                <property name="presenterFactory" ref="crudPresenterFactory" />
                <property name="securityViewFactory" ref="securityViewFactory" />
                <property name="securityPresenterFactory" ref="securityPresenterF
                <property name="crudFactory" ref="crudFactory" />
        </bean>

        <bean id="crudFactory" class="ar.com.oxen.nibiru.crud.manager.jpa.JpaCru
                <property name="entityManager">
                        <bean factory-ref="entityManagerFactory" factory-method=
                </property>
                <property name="wrapperFactory" ref="wrapperFactory" />
                <property name="authorizationService" ref="authorizationService"
                <property name="transactionTemplate" ref="transactionTemplate" />
        </bean>
</blueprint>
```

15.3.2 Automatic login module

If your application doesn't have security requirements, you can use the ar.com.oxen.nibiru.security.autologin module. It performs automatic login and provides dummy implementation for security services.

Just include this module into the installation. No other security module/implementation is required.

15.4 Remote access

The ar.com.oxen.nibiru.security.rpc module provides classes for exposing security services over a network.

The RpcAuthenticationService allows exposing a remote authentication service:

```
package ar.com.oxen.nibiru.security.rpc;

/**
 * Remote authentication service.
 *
 */
```

```
public interface RpcAuthenticationService {
        /**
         * Performs a login.
         *
         * @param loginDto
         *            A DTO with login request information
         * @return A DTO with user information
         */
        UserDto login(LoginDto loginDto);
}
```

16 Transaction management

There are transaction demarcation mechanisms (using AOP) which are not intrusive. Where possible, such mechanisms are used. Its implementation depends on the platform. For example, when using Blueprint, you could use Aries JTA integration. When running on a non-OSGi environment (with Spring), you can choose a PlatformTransactionManager implementation.

An specific API was defined in order to provide programmatic transaction management. It is very limited, just to satisfy Nibiru requirements.

Currently, the only interface is TransactionTemplate:

```
package ar.com.oxen.nibiru.transaction.api;

import javax.persistence.EntityManager;

/**
 * A template for making calls inside a transaction.
 *
 */
public interface TransactionTemplate {
        /**
         * Executes the callback inside a transaction, joining the EntityManager
         *
         * @param entityManager
         *            The entity manager
         * @param callback
         *            The callback
         * @return The value returned by the callback
         */
        <T> T execute(EntityManager entityManager, TransactionCallback<T> callba
}
```

which allows running a callback inside a transaction (with "required" semantics: if a transaction is active, it is used, otherwise, a new one is created).

The ar.com.oxen.nibiru.transaction.jta bundle provides a JTA implementation for such template. It receives the UserTransaction (which is exposed as an OSGi service using Aries/Karaf).

17 Internationalization

The ar.com.oxen.nibiru.i18n.api project contains interfaces for internationalization. There are 3 main services:

1. LocaleHolder: Used to read or write the user's Locale.
2. MessageSource: Used to get messages by key (with parameters).
3. MessageProvider: Used to provide message querying using a key and a Locale. This division was made so that each module can provide its own MessageProvider. Typically there will be a MessageSource implementation that consolidates them.

The 3 interfaces are very simple, as you can see.

- LocaleHolder:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * Service used to access the user locale.
 *
 */
public interface LocaleHolder {
        /**
         * Gets the user locale.
         *
         * @return The locale
         */
        Locale getLocale();

        /**
         * Sets the user locale.
         *
         * @param newLocale
         *             The locale
```

```
         */
        void setLocale(Locale newLocale);
}
```

- MessageSource:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * Service for accessing i18n messages. Typically a view from a module will
 * access this service. Internally, implementation of this module should acc
 * the current user locale with {@link LocaleHolder} and delegate on N
 * {@link MessageProvider}s in order to look for the searched message.
 */
public interface MessageSource {
        /**
         * Gets a i18n message
         *
         * @param code
         *            The message code
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not f
         *         returns the code.
         */
        String getMessage(String code, Object... args);

        /**
         * Returns a 18n message
         *
         * @param code
         *            The message code
         * @param locale
         *            The locale
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not f
         *         the code is returned.
         */
        String getMessage(String code, Locale locale, Object... args);

        /**
         * Gets a i18n message
         *
```

```
         * @param code
         *            The message code
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not f
         *         returns null.
         */
        String findMessage(String code, Object... args);

        /**
         * Returns a 18n message
         *
         * @param code
         *            The message code
         * @param locale
         *            The locale
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not f
         *         null is returned.
         */
        String findMessage(String code, Locale locale, Object... args);
}
```

- MessageProvider:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * A message provider. This interface is provided in order to allow i18n
 * modularity. Each module could provide its own MessageProvider. All the
 * MessageProviders would be consolidated by a single, generic
 * {@link MessageSource}.
 */
public interface MessageProvider {
        /**
         * Returns a 18n message
         *
         * @param code
         *            The message code
         * @param locale
         *            The locale
         * @param args
         *            The message arguments
```

```
         * @return The translated an parsed message. If the message is not f
         *         null is returned.
         */
        String getMessage(String code, Locale locale, Object... args);
}
```

17.1 Generic implementation

The ar.com.oxen.nibiru.i18n.generic project contains an generic MessageSource implementation which is injected with LocaleHolder and a list of MessageProviders. OSGi Blueprint can inject a MessageProvider service list that is updated dynamically according to the availability of new instances of these services. This project also contains a MessageProvider implementation based on ResoruceBundle.

17.2 Session integration

The ar.com.oxen.nibiru.i18n.session project has a LocaleHolder implementation that stores the locale in the Nibiru session.

18 Validation

The ar.com.oxen.nibiru.validation.api project defines the validation API. It includes two main interfaces.

- Validator, which represents a component that can perform a validation:

```
package ar.com.oxen.nibiru.validation.api;

/**
 * Interface for classes that can perform validation.
 *
 * @param <T>
 *            Validated object data type.
 */
public interface Validator<T> {
        /**
         * Validates an object.
         *
         * @param object
         *            The object to be valdiated.
         * @throws ValidationException
         *             If the validation is not met.
```

```
         */
        void validate(T object) throws ValidationException;
}
```

- Validatable, representing a component which can have validators associated to it:

```
package ar.com.oxen.nibiru.validation.api;

/**
 * Something that can be validated.
 *
 * @param <T>
 *            Validated object data type.
 */
public interface Validatable<T> {
        /**
         * Adss a validator.
         *
         * @param validator
         *            The validator
         */
        void addValidator(Validator<T> validator);

        /**
         * Removes a validator.
         *
         * @param validator
         *            The validator
         */
        void removeValidator(Validator<T> validator);

        /**
         * Validates the validatable data.
         *
         * @throws ValidationException
         *             If the validation is not met.
         */
        void validate() throws ValidationException;
}
```

18.1 Generic validators

The ar.com.oxen.nibiru.validation.generic project provides validators that can be reused among projects.

- NotEmptyValidator, that checks against null or “” value:

```
package ar.com.oxen.nibiru.validation.generic;

import ar.com.oxen.nibiru.validation.api.ValidationException;
import ar.com.oxen.nibiru.validation.api.Validator;

/**
 * Validatro for using on required fields.
 * The validated object:
 *  - Must not be null
 *  - It String representation must not be "" neither spaces.
 */
public class NotEmptyValidator implements Validator<Object> {
        private String errorCode = "required";

        public NotEmptyValidator() {
                super();
        }

        public NotEmptyValidator(String errorCode) {
                super();
                this.errorCode = errorCode;
        }

        @Override
        public void validate(Object object) throws ValidationException {
                if (object == null || object.toString().trim().equals("")) {
                        throw new ValidationException(errorCode);
                }
        }
}
```

- RegexpValidator, which checks the value against a regular expression:

```
package ar.com.oxen.nibiru.validation.generic;

import java.util.regex.Pattern;

import ar.com.oxen.nibiru.validation.api.ValidationException;
import ar.com.oxen.nibiru.validation.api.Validator;

/**
 * Regexp-based validator. Validated must be an String
 */
public class RegexpValidator implements Validator<String> {
        private Pattern pattern;
```

```
        private String errorCode;

        public RegexpValidator(String regexp, String errorCode) {
                super();
                this.pattern = Pattern.compile(regexp);
                this.errorCode = errorCode;
        }

        @Override
        public void validate(String object) throws ValidationException {
                if (!pattern.matcher(object).matches()) {
                        throw new ValidationException(errorCode);
                }
        }
}
```

19 CRUD

CRUD module (Create, Read, Update and Delete) aims to facilitate the generation of funcionality of this type.

The functionality of this module is distributed across multiple bundles. It can be grouped into 2 layers.

19.1 Persistence services

The required interfaces for exposing persistence services are found in the ar.com.oxen.nibiru.crud.manager.api project.

The main interface is CrudManager, which provides the necessary methods to dynamically generate an CRUD screen. In other words, the idea is to have a CrudManager by each entity on which you want to build a CRUD.

```
package ar.com.oxen.nibiru.crud.manager.api;

import java.util.List;

/**
 * Service for managing CRUD over entities.
 *
 * @param <T>
 *            The crud entity type.
 */
public interface CrudManager<T> {
```

```
    /**
     * Returns the entity type name.
     *
     * The name identifies the kind of entity being handled. This is useful,
     * example, in order to determine if a given entity is compatible with a
     * crud manager.
     *
     * @return The type name.
     */
    String getEntityTypeName();

    /**
     * Gets the fields to be shown in the entity list.
     *
     * @return A list with the fields
     */
    List<CrudField> getListFields();

    /**
     * Reads all the entities.
     *
     * @return A list with the entities
     */
    List<CrudEntity<T>> findAll();

    /**
     * Finds an entity by its ID.
     *
     * @return The entity
     */
    CrudEntity<T> findById(Object id);

    /**
     * Reads entities filtering by a given field. Useful for parent-child
     * relations
     *
     * @return A list with the entities
     */
    List<CrudEntity<T>> findByfield(String field, Object value);

}
```

CRUD module is designed for handling various types of entities. Unlike a typical CRUD generator, where screens are generated to manage tables in a database or on beans, Nibiru CRUD adds a level of indirection. This allows you to create persistence service implementations providing access to beans JPA, business

process instances, and so on.

The interfaces used to achieve this level of abstraction are CrudEntity (representing an entity that is being edited) and CrudField (which represents a field of such entity).

```
package ar.com.oxen.nibiru.crud.manager.api;

import java.util.List;

/**
 * Represents an entity instance. This interface is used in order to hide entity
 * implementation. This way, CRUD engine could work over Java beans, BPM
 * processes, etc.
 *
 * @param <T>
 */
public interface CrudEntity<T> {
        /**
         * Reads the id value.
         *
         * @return The id
         */
        Object getId();

        /**
         * Gets the fields to be shown in the entity form.
         *
         * @return A list with the fields
         */
        List<CrudField> getFormFields();


        /**
         * Reads a field value.
         *
         * @param field
         *              The field
         * @return The value
         */
        Object getValue(CrudField field);

        /**
         * Reads a field value.
         *
         * @param fieldName
         *              The field name
```

```
     * @return The value
     */
    Object getValue(String fieldName);

    /**
     * Writes a field value
     *
     * @param field
     *            The field
     * @param value
     *            The value
     */
    void setValue(CrudField field, Object value);

    /**
     * Writes a field value
     *
     * @param fieldName
     *            The field name
     * @param value
     *            The value
     */
    void setValue(String fieldName, Object value);

    /**
     * Gets the wrapped object.
     *
     * @return The entity object
     */
    T getEntity();

    /**
     * Returns the entity type name.
     *
     * The name identifies the kind of entity being handled. This is useful,
     * example, in order to determine if a given entity is compatible with a
     * crud manager.
     *
     * @return The type name.
     */
    String getEntityTypeName();

    /**
     * Returns the available values for a given field (for example, for usin
     * a combo box or a list select)
     *
```

```
         * @param field
         *             The field
         * @return An iterable for the values
         */
        Iterable<?> getAvailableValues(CrudField field);

        /**
         * Returns the available values for a given field (for example, for usin
         * a combo box or a list select)
         *
         * @param fieldName
         *             The field name
         * @return An iterable for the values
         */
        Iterable<?> getAvailableValues(String fieldName);
}

package ar.com.oxen.nibiru.crud.manager.api;

/**
 * Represents a field on a {@link CrudEntity}.
 *
 */
public interface CrudField {
        /**
         * @return The field name
         */
        String getName();

        /**
         * @return The field class
         */
        Class<?> getType();

        /**
         * @return Information for showing the field in a list.
         */
        ListInfo getListInfo();

        /**
         * @return Information for showing the field in a form.
         */
        FormInfo getFormInfo();

        /**
         * Information for showing the field in a list.
```

```
 */
interface ListInfo {
        /**
         * Determines a fixed width for the field column.
         *
         * @return The column width
         */
        int getColumnWidth();

}

/**
 * Information for showing the field in a form.
 */
interface FormInfo {
        String GENERAL_TAB = "general";

        /**
         * Determines how the field should be represented (for example,
         * form).
         *
         * @return An element of widget type enumeration
         */
        WidgetType getWidgetType();

        /**
         * @return True if the field can't be modified
         */
        boolean isReadonly();

        /**
         * Determines how many characters can be set on the field. Appli
         * to widgets which holds Strings.
         *
         * @return The maximum length
         */
        int getMaxLength();

        /**
         * Returns the tab name where the widget must be shown.
         *
         * @return The tab name
         */
        String getTab();
}
}
```

WidgetType enumerates the ways in which a field can be shown:

```
package ar.com.oxen.nibiru.crud.manager.api;

public enum WidgetType {
        TEXT_FIELD,
        PASSWORD_FIELD,
        TEXT_AREA,
        DATE_FIELD,
        TIME_FIELD,
        CHECK_BOX,
        COMBO_BOX,
        MULTISELECT
}
```

The abstraction would not be complete if the actions to be performed on the entities weren't not configurable. To this end the CrudAction interface was created.

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * Represents an action that can be applied on a CRUD. Abstracting the actions
 * allows the CRUD implementations to provide extra actions. This way, actions
 * are not limited to create, read, update an delete (so the module shouldn't be
 * called CRUD!!!), but can add action such as approve, reject, start, stop,
 * etc. In some cases, the action can require no entity (for example, "new"). In
 * other cases, it would be mandatory applying the action over an specific
 * {@link CrudEntity} ("edit", for example).
 *
 */
public interface CrudAction {
        String NEW = "new";
        String DELETE = "delete";
        String EDIT = "edit";
        String UPDATE = "update";

        /**
         * Gets the action name.
         *
         * @return The name
         */
        String getName();

        /**
         * Indicates if the action must be performed over an {@link CrudEntity}.
         *
```

```
     * @return True if a {@link CrudEntity} is required
     */
    boolean isEntityRequired();

    /**
     * Indicates if a user confirmation must be presented before performing
     * action.
     *
     * @return True if confirmation must be presented
     */
    boolean isConfirmationRequired();

    /**
     * Indicates if the action must be shown in list window.
     *
     * @return True if it must be shown
     */
    boolean isVisibleInList();

    /**
     * Indicates if the action must be shown in form window.
     *
     * @return True if it must be shown
     */
    boolean isVisibleInForm();

    /**
     * Indicates if the action modifies the entity.
     *
     * @return True if entity is modified
     */
    boolean modifiesEntity();

    String[] getAllowedRoles();
}
```

In this way the actions are not limited to create, read, update and delete, but they are extensible. A workflow engine could, for example, display actions such as "approve" or "reject."

The getAllowedRoles method indicates the required roles in order to execute the action. Such roles are validated against security services. If no role is specified (or null is returned), no validation is done (so everybody can execute the action).

In order to make the CRUD modular, the actions to perform on an entity are not provided directly by the CrudManager, but using the extension point mechanism. The interface CrudActionExtension allows implementing extensions that

add different possible actions to perform on an entity.

```
package ar.com.oxen.nibiru.crud.manager.api;

import java.util.List;

/**
 * Extension used to add actions to CRUD.
 *
 * @param <T>
 *            The {@link CrudEntity} type
 */
public interface CrudActionExtension<T> {
        /**
         * Gets global actions provided by this extension.
         *
         * @return A list with the actions
         */
        List<CrudAction> getGlobalActions();

        /**
         * Gets entity actions provided by this extension.
         *
         * @return A list with the actions
         */
        List<CrudAction> getEntityActions(CrudEntity<T> entity);

        /**
         * Performs an action over a given entity. The action can create/update
         * entity. In that case, such entity is returned, otherwise it returns n
         * When a created/updated entity is returned, the CRUD should open a form
         * order to edit it. This can be useful, for example, for BPM
         * implementations that jumps from an activity to another.
         *
         * @param action
         *            The action entity)
         * @return The created/updated entity
         */
        CrudEntity<?> performGlobalAction(CrudAction action);

        /**
         * Performs an action over a given entity. The action can create/update
         * entity. In that case, such entity is returned, otherwise it returns n
         * When a created/updated entity is returned, the CRUD should open a form
         * order to edit it. This can be useful, for example, for BPM
         * implementations that jumps from an activity to another.
```

```
     *
     * @param action
     *             The action
     * @param entity
     *             The entity (it can be null if the action doesn't require a
     *             entity)
     * @return The created/updated entity
     */
    CrudEntity<?> performEntityAction(CrudAction action, CrudEntity<T> entit

    /**
     * @return The allowed roles
     */
    String[] getAllowedRoles();
}
```

The ar.com.oxen.nibiru.crud.manager.jpa bundle contains implementations based on JPA. It relies on ar.com.oxen.nibiru.crud.bean and ar.com.oxen.nibiru.crud.utils classes. Where possible, it uses JPA information and reflection to return the information required for CRUD. Where not possible, it uses ar.com.oxen.nibiru.crud.bean based on annotations.

19.1.1 Events

The CRUD API provides some common use events. They are intended to be used when communicating the different CRUD components through the event bus.

The ManageCrudEntitiesEvent can be used in order to notify that administration of entities of a given type is required. This event is tipically fired from a menu.

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * This is a generic event class for triggering entities management. The topic
 * should be used in order to identify the entity to be managed.
 */
public class ManageCrudEntitiesEvent {
}
```

The EditCrudEntityEvent indicates that a given entity must be edited This tipically will open a CRUD form.

```
package ar.com.oxen.nibiru.crud.manager.api;

import ar.com.oxen.nibiru.conversation.api.Conversation;
```

```
public class EditCrudEntityEvent<T> {
        private CrudEntity<T> entity;
        private Conversation conversation;

        public EditCrudEntityEvent(CrudEntity<T> entity, Conversation conversatio
                super();
                this.entity = entity;
                this.conversation = conversation;
        }

        public CrudEntity<T> getCrudEntity() {
                return entity;
        }

        public Conversation getConversation() {
                return conversation;
        }
}
```

When editing is finished, a ModifiedCrudEntityEvent can be fired in order to notify that such instance has been modified. For example, the CRUD list presenter listens to this event in order to refresh the list.

```
package ar.com.oxen.nibiru.crud.manager.api;

public class ModifiedCrudEntityEvent {
        private Object id;

        public ModifiedCrudEntityEvent(Object id) {
                super();
                this.id = id;
        }

        public Object getId() {
                return id;
        }
}
```

Finally, a ManageChildCrudEntitiesEvent can be fired in order to activate a CRUD for dependant entities (in a parent-child relationship).

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * This is a generic event class for triggering entities management related to a
 * parent. The topic should be used in order to identify the entity to be
```

```
 * managed.
 */
public class ManageChildCrudEntitiesEvent {
        private String parentField;
        private Object parentEntity;

        public ManageChildCrudEntitiesEvent(String parentField, Object parentEnt
                super();
                this.parentField = parentField;
                this.parentEntity = parentEntity;
        }

        public String getParentField() {
                return parentField;
        }

        public Object getParentEntity() {
                return parentEntity;
        }
}
```

19.2 User interface services

The ar.com.oxen.nibiru.crud.ui.api project contains interfaces for CRUD views and presenters.

These interfaces must be instantiated by a presenter factory implementation:

```
package ar.com.oxen.nibiru.crud.ui.api;

import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.crud.manager.api.EditCrudEntityEvent;
import ar.com.oxen.nibiru.crud.ui.api.form.CrudFormView;
import ar.com.oxen.nibiru.crud.ui.api.list.CrudListView;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;

/**
 * CRUD presenter factory.
 */
public interface CrudPresenterFactory {
        /**
         * Builds a presenter for CRUD list.
         *
         * @param crudManager
         *              The CRUD manager
         * @return The presenter
```

```
         */
        <T> Presenter<CrudListView> buildListPresenter(CrudManager<T> crudManage

        /**
         * Builds a presenter for CRUD list which is filtered by a parent value.
         *
         * @param crudManager
         *            The CRUD manager
         * @param parentField
         *            The field used in order to filter the parent value.
         * @param parentValue
         *            The parent value.
         * @return The presenter
         */
        <T> Presenter<CrudListView> buildListPresenter(CrudManager<T> crudManage
                    String parentField, Object parentValue);

        /**
         * Builds a presenter for CRUD form.
         *
         * @param crudManager
         *            The CRUD manager
         * @return The presenter
         */
        <T> Presenter<CrudFormView> buildFormPresenter(CrudManager<T> crudManage
                    EditCrudEntityEvent<T> event);
}
```

and a view factory:

```
package ar.com.oxen.nibiru.crud.ui.api;

import ar.com.oxen.nibiru.crud.ui.api.form.CrudFormView;
import ar.com.oxen.nibiru.crud.ui.api.list.CrudListView;

/**
 * CRUD presenter factory.
 */
public interface CrudViewFactory {
        String I18N_FIELD_PREFIX = "ar.com.oxen.nibiru.crud.field.";
        String I18N_ACTION_PREFIX = "ar.com.oxen.nibiru.crud.action.";
        String I18N_ENTITY_PREFIX = "ar.com.oxen.nibiru.crud.entity.";
        String I18N_TAB_PREFIX = "ar.com.oxen.nibiru.crud.tab.";
        String I18N_ERROR_PREFIX = "ar.com.oxen.nibiru.crud.error.";

        /**
         * Builds the view for CRUD list.
```

```
     *
     * @return The view
     */
    CrudListView buildListView();

    /**
     * Builds the view for CRUD form.
     *
     * @return The view
     */
    CrudFormView buildFormView();
}
```

There is a generic implementation in the ar.com.oxen.nibiru.crud.ui.generic project.

19.3 Utilities

The ar.com.oxen.nibiru.crud.utils bundle contains generic utility classes for creating CRUDs. This includes:

- Simple implementations for CrudField and CrudAction.
- Common action extensions.
- A base class for CRUD modules configuration (AbstractCrudModuleConfigurator).

The AbstractCrudModuleConfigurator class provides the following methods:

- addCrud: Adds a top-level CRUD, which are started from application menu. The method registers the extension points for menu and actions. Also, it registers event bus listeners for navigation.
- addChildCrud: Adds a child CRUD, which is fired from a parent CRUD contextual menu. In a similar way, it registers the appropiate extensions and listeners.

```
package ar.com.oxen.nibiru.crud.utils;

import java.util.LinkedList;
import java.util.List;

import ar.com.oxen.commons.eventbus.api.EventHandler;
import ar.com.oxen.nibiru.crud.manager.api.CrudActionExtension;
import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.crud.manager.api.EditCrudEntityEvent;
```

```
import ar.com.oxen.nibiru.crud.manager.api.ManageChildCrudEntitiesEvent;
import ar.com.oxen.nibiru.crud.manager.api.ManageCrudEntitiesEvent;
import ar.com.oxen.nibiru.crud.ui.api.CrudPresenterFactory;
import ar.com.oxen.nibiru.crud.ui.api.CrudViewFactory;
import ar.com.oxen.nibiru.module.utils.AbstractModuleConfigurator;
import ar.com.oxen.nibiru.ui.api.extension.MenuItemExtension;
import ar.com.oxen.nibiru.ui.utils.extension.SimpleMenuItemExtension;
import ar.com.oxen.nibiru.ui.utils.mvp.SimpleEventBusClickHandler;

public abstract class AbstractCrudModuleConfigurator extends
        AbstractModuleConfigurator<CrudViewFactory, CrudPresenterFactory
    private List<EventHandler<?>> registeredHandlers = new LinkedList<EventH

    int menuPos = 0;

    /**
     * Adds a CRUD menu
     */
    protected <K> void addCrudMenu(String menuName, String parentMenuExtensi
            CrudManager<K> crudManager) {
        this.registerMenu(menuName, parentMenuExtension, crudManager);
    }

    /**
     * Adds a CRUD menu with allowed roles
     */
    protected <K> void addCrudMenu(String menuName, String parentMenuExtensi
            CrudManager<K> crudManager, String[] allowedRoles) {
        this.registerMenu(menuName, parentMenuExtension, crudManager,
                allowedRoles);
    }

    /**
     * Adds a CRUD without a menu option
     */
    protected <K> void addCrud(CrudManager<K> crudManager,
            CrudActionExtension<K> crudActionExtension) {
        this.registerManageEntityEvent(crudManager);
        this.registerActions(crudManager, crudActionExtension);
        this.registerEditEntityEvent(crudManager);
    }

    /**
     * Adds a CRUD with a menu option
     */
    protected <K> void addCrudWithMenu(String menuName,
```

```
                String parentMenuExtension, CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension) {
        this.addCrudWithMenu(menuName, parentMenuExtension, crudManager,
                        crudActionExtension, null);
}

/**
 * Adds a CRUD with a menu option and allowed roles
 */
protected <K> void addCrudWithMenu(String menuName,
                String parentMenuExtension, CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension, String[] allo
        this.addCrudMenu(menuName, parentMenuExtension, crudManager,
                        allowedRoles);
        this.addCrud(crudManager, crudActionExtension);
}

/**
 * Adds a child menu CRUD menu option
 */
protected <T> void addChildCrudMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager) {

        this.addChildCrudMenu(menuName, parentCrudManager, parentField,
                        childCrudManager, null);
}

/**
 * Adds a child menu CRUD menu option and allowed roles
 */
protected <T> void addChildCrudMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager, String[] allowedRoles)

        this.registerManageChildrenAction(menuName, parentCrudManager,
                        childCrudManager, parentField, allowedRoles);
}

/**
 * Adds a child menu CRUD without a menu option
 */
protected <T> void addChildCrud(CrudManager<?> parentCrudManager,
                CrudManager<T> childCrudManager,
                CrudActionExtension<T> childCrudActionExtension) {
```

```
        this.registerActions(childCrudManager, childCrudActionExtension)
        this.registerManageChildEntitiesEvent(parentCrudManager,
                        childCrudManager);
        this.registerEditEntityEvent(childCrudManager);
}

/**
 * Adds a child menu CRUD with a menu option
 */
protected <T> void addChildCrudWithMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager,
                CrudActionExtension<T> childCrudActionExtension) {

        this.addChildCrudMenu(menuName, parentCrudManager, parentField,
                        childCrudManager);
        this.addChildCrud(parentCrudManager, childCrudManager,
                        childCrudActionExtension);
}

/**
 * Adds a child menu CRUD with a menu option and allowed roles
 */
protected <T> void addChildCrudWithMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager,
                CrudActionExtension<T> childCrudActionExtension,
                String[] allowedRoles) {

        this.addChildCrudMenu(menuName, parentCrudManager, parentField,
                        childCrudManager, allowedRoles);
        this.addChildCrud(parentCrudManager, childCrudManager,
                        childCrudActionExtension);
}

@Override
public void shutdown() {
        super.shutdown();
        for (EventHandler<?> handler : this.registeredHandlers) {
                this.getEventBus().removeHandler(handler);
        }
}

protected void registerMenu(String menuName, String parentMenuExtension,
                CrudManager<?> crudManager) {
        this.registerExtension(
```

```
                        new SimpleMenuItemExtension(menuName, menuPos++,
                                new SimpleEventBusClickHandler(t
                                        ManageCrudEntiti

                        parentMenuExtension, MenuItemExtension.class);
    }

    protected void registerMenu(String menuName, String parentMenuExtension,
                    CrudManager<?> crudManager, String[] allowedRoles) {
            this.registerExtension(
                        new SimpleMenuItemExtension(menuName, menuPos++,
                                new SimpleEventBusClickHandler(t
                                        ManageCrudEntiti

                        parentMenuExtension, MenuItemExtension.class);
    }

    protected <K> void registerActions(CrudManager<K> crudManager,
                    CrudActionExtension<K> crudActionExtension) {
            this.registerExtension(crudActionExtension,
                        crudManager.getEntityTypeName(), CrudActionExten
    }

    protected <K> void registerManageChildrenAction(String menuName,
                    CrudManager<?> parentCrudManager,
                    final CrudManager<?> childCrudManager, String parentFiel
            this.registerManageChildrenAction(menuName, parentCrudManager,
                        childCrudManager, parentField, null);
    }

    protected <K> void registerManageChildrenAction(String menuName,
                    CrudManager<?> parentCrudManager,
                    final CrudManager<?> childCrudManager, String parentFiel
                    String[] allowedRoles) {
            this.registerExtension(new ManageChildrenCrudActionExtension<Obj
                        menuName, parentField, childCrudManager.getEntity
                        this.getEventBus(), allowedRoles), parentCrudMan
                        .getEntityTypeName(), CrudActionExtension.class)
    }

    protected void registerManageEntityEvent(final CrudManager<?> crudManage
            this.addEventHandler(ManageCrudEntitiesEvent.class,
                        new EventHandler<ManageCrudEntitiesEvent>() {

                                @Override
                                public void onEvent(ManageCrudEntitiesEv
```

```
                                        activate(
                                                getViewFactory()
                                                getPresenterFact

                                        }
                                }, crudManager.getEntityTypeName());
        }

        protected <X> void registerEditEntityEvent(final CrudManager<X> crudMana
                Class<EditCrudEntityEvent<X>> eventClass = getEditCrudEntityEven
                this.addEventHandler(eventClass,
                                new EventHandler<EditCrudEntityEvent<X>>() {

                                        @Override
                                        public void onEvent(EditCrudEntityEvent<
                                                activate(
                                                getViewFactory()
                                                getPresenterFact

                                        }
                                }, crudManager.getEntityTypeName());
        }

        @SuppressWarnings({ "rawtypes", "unchecked" })
        private <X> Class<EditCrudEntityEvent<X>> getEditCrudEntityEventClass()
                return (Class) EditCrudEntityEvent.class;
        }

        protected void registerManageChildEntitiesEvent(
                        CrudManager<?> parentCrudManager,
                        final CrudManager<?> childCrudManager) {
                this.addEventHandler(ManageChildCrudEntitiesEvent.class,
                                new EventHandler<ManageChildCrudEntitiesEvent>()

                                        @Override
                                        public void onEvent(ManageChildCrudEntit
                                                activate(
                                                getViewFactory()
                                                getPresenterFact



                                        }
                                }, childCrudManager.getEntityTypeName());
        }
```

```
        private <T> void addEventHandler(Class<T> eventClass,
                        EventHandler<T> handler, String topic) {
                this.registeredHandlers.add(handler);
                this.getEventBus().addHandler(eventClass, handler, topic);
        }
}
```

19.4 Bean-based CRUDs

The ar.com.oxen.nibiru.crud.bean project contains utility classes for CRUD implementations that use beans, like an implementation of CrudEntity that delegates to a bean (through BeanWrapper from Java Oxen Commons). Also, it contains annotations which are useful in order to to parametrize the CRUD directly on the bean.

For example, the following class shows some bean annotations:

```
package ar.com.oxen.nibiru.sample.domain;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import ar.com.oxen.nibiru.crud.bean.annotation.Action;
import ar.com.oxen.nibiru.crud.bean.annotation.Actions;
import ar.com.oxen.nibiru.crud.bean.annotation.Filter;
import ar.com.oxen.nibiru.crud.bean.annotation.Show;
import ar.com.oxen.nibiru.crud.bean.annotation.Widget;
import ar.com.oxen.nibiru.crud.manager.api.CrudAction;
import ar.com.oxen.nibiru.crud.manager.api.WidgetType;

@Entity
@Actions({
                @Action(name = CrudAction.NEW, requiresEntity = false, showInForm
                @Action(name = CrudAction.EDIT, requiresEntity = true, showInForm
                @Action(name = CrudAction.UPDATE, requiresEntity = true, showInF
                @Action(name = CrudAction.DELETE, requiresEntity = true, showInFo
@Filter("authz.isCallerInRole('ar.com.oxen.nibiru.security.role.Administrator')␣
public class Student {
        @Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Show(order = 0)
@Widget(type = WidgetType.TEXT_FIELD, readonly = true)
private Integer id;

@Column
@Show(order = 10)
private String name;

@Column
@Show(order = 15, inList = true)
private Boolean active;

@ManyToMany(mappedBy = "students", cascade = CascadeType.ALL)
@Show(order = 30, inList = false)
@Widget(type = WidgetType.MULTISELECT, tab = "courses")
private Set<Course> courses;

public Integer getId() {
        return id;
}

public void setId(Integer id) {
        this.id = id;
}

public Boolean getActive() {
        return active;
}

public void setActive(Boolean active) {
        this.active = active;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public Set<Course> getCourses() {
        return courses;
}
```

```
        public void setCourses(Set<Course> courses) {
                /*
                 * OpenJPA updates in cascade just one side on a bidirectional r
                 * http://openjpa.apache.org/builds/1.0.1/apache-openjpa-1.0.1/d
                 */
                for (Course course : this.courses) {
                        if (!courses.contains(course)) {
                                course.getStudents().remove(this);
                        }
                }
                for (Course course : courses) {
                        course.getStudents().add(this);
                }
                this.courses = courses;
        }

        @Override
        public String toString() {
                return this.name;
        }

        @Override
        public int hashCode() {
                final int prime = 31;
                int result = 1;
                result = prime * result + ((id == null) ? 0 : id.hashCode());
                return result;
        }

        @Override
        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (obj == null)
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                Student other = (Student) obj;
                if (id == null) {
                        if (other.id != null)
                                return false;
                } else if (!id.equals(other.id))
                        return false;
                return true;
        }
}
```

- @Actions/@Action: Defines which actions can be performed on the entity or just on the CRUD window.
- @Filter: Allows an arbitrary filtering expression, wich can be evaluated and passed to the CrudManager (for example, for using in a JPA query). This is useful, for example, when defining row-level security.
- @Show: Determines how and where the field is shown.
- @Widget: Provides information about how the UI widget must be generated.

19.5 Validation

Validation over CRUD fields can be done by exposing a Validator as an extension.

The extension point name must be built with the entiy name, appending a dot and the name of the field to be validated. For example:

```
...

this.registerExtension(new NotEmptyValidator(),
                       Subject.class.getName() +".description",
                       Validator.class);

...
```

20 Reports

The report API is defined in the ar.com.oxen.nibiru.report.api project. It includes just one interface:

```
package ar.com.oxen.nibiru.report.api;

import java.io.InputStream;
import java.util.Map;

public interface Report {
        String EXTENSION_POINT_NAME = "ar.com.oxen.nibiru.reports";

        String getName();

        Iterable<String> getFormats();

        Iterable<ParameterDefinition> getParameterDefinitions();
```

```
        InputStream render(String format, Map<String, Object> parameters);

        interface ParameterDefinition {
                String getName();

                Class<?> getType();
        }
}
```

Such interface must be implemented by any report, regardless the used engine. As the report will be usually exposed as an extension, a name for the corresponding extension point is provided.

20.1 BIRT implementation

The ar.com.oxen.nibiru.report.birt project provides a report implementation using BIRT:

```
package ar.com.oxen.nibiru.report.birt;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import java.util.Map;

import org.eclipse.birt.report.engine.api.EngineConfig;
import org.eclipse.birt.report.engine.api.EngineConstants;
import org.eclipse.birt.report.engine.api.EngineException;
import org.eclipse.birt.report.engine.api.HTMLRenderOption;
import org.eclipse.birt.report.engine.api.IPDFRenderOption;
import org.eclipse.birt.report.engine.api.IParameterDefn;
import org.eclipse.birt.report.engine.api.IRenderOption;
import org.eclipse.birt.report.engine.api.IReportEngine;
import org.eclipse.birt.report.engine.api.IReportRunnable;
import org.eclipse.birt.report.engine.api.IRunAndRenderTask;
import org.eclipse.birt.report.engine.api.PDFRenderOption;
import org.eclipse.birt.report.engine.api.RenderOption;
import org.eclipse.birt.report.engine.api.ReportEngine;
```

```
import ar.com.oxen.nibiru.report.api.Report;

public class BirtReport implements Report {
        private IReportEngine engine;
        private IReportRunnable design;

        public BirtReport(String file) {
                super();
                try {
                        engine = new ReportEngine(new EngineConfig());
                        this.design = engine.openReportDesign(this
                                        .getReporFileInputStream(file));
                } catch (EngineException e) {
                        throw new BirtReportException(e);
                }
        }

        @Override
        public String getName() {
                return this.design.getDesignInstance().getDisplayName();
        }

        @Override
        public Iterable<String> getFormats() {
                return Arrays.asList(new String[] { "pdf", "html" });
        }

        @Override
        public Iterable<ParameterDefinition> getParameterDefinitions() {
                @SuppressWarnings("unchecked")
                Collection<IParameterDefn> parameterDefns = engine
                                .createGetParameterDefinitionTask(design).getPar
                                                false);
                List<ParameterDefinition> parameters = new ArrayList<Report.Para
                                parameterDefns.size());

                for (IParameterDefn parameterDefn : parameterDefns) {
                        parameters.add(new IParameterDefnAdapter(parameterDefn))
                }

                return parameters;
        }

        @Override
        public InputStream render(final String format,
```

```
                final Map<String, Object> parameters) {
        try {
                final PipedOutputStream output = new PipedOutputStream()
                InputStream input = new PipedInputStream(output);

                final ClassLoader classLoader = Thread.currentThread()
                                .getContextClassLoader();

                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                render(format, parameters, output, classl
                        }
                }).start();

                return input;
        } catch (IOException e) {
                throw new BirtReportException(e);
        }
}

@SuppressWarnings("unchecked")
private void render(String format, Map<String, Object> parameters,
                OutputStream output, ClassLoader classLoader) {
        try {
                /* Create task to run and render the report */
                IRunAndRenderTask task = design.getReportEngine()
                                .createRunAndRenderTask(design);

                /* Set parent classloader for engine */
                task.getAppContext().put(
                                EngineConstants.APPCONTEXT_CLASSLOADER_K

                for (Map.Entry<String, Object> entry : parameters.entryS
                        task.setParameterValue(entry.getKey(), entry.get
                }

                final IRenderOption options = new RenderOption();
                options.setOutputFormat(format);

                options.setOutputStream(output);

                if (options.getOutputFormat().equalsIgnoreCase("html"))
                        final HTMLRenderOption htmlOptions = new HTMLRe
                                        options);
                        htmlOptions.setImageDirectory("img");
```

```
                htmlOptions.setHtmlPagination(false);
                htmlOptions.setHtmlRtLFlag(false);
                htmlOptions.setEmbeddable(false);
                htmlOptions.setSupportedImageFormats("PNG");
            } else if (options.getOutputFormat().equalsIgnoreCase("p
                final PDFRenderOption pdfOptions = new PDFRender
                pdfOptions.setOption(IPDFRenderOption.PAGE_OVERFl
                        IPDFRenderOption.FIT_TO_PAGE_SIZ
                pdfOptions.setOption(IPDFRenderOption.PAGE_OVERFl
                        IPDFRenderOption.OUTPUT_TO_MULTI
            }

            task.setRenderOption(options);

            // run and render report
            task.run();

            task.close();

            output.flush();
            output.close();
        } catch (EngineException e) {
            throw new BirtReportException(e);
        } catch (IOException e) {
            throw new BirtReportException(e);
        }
    }

    private InputStream getReporFileInputStream(String file) {
        InputStream reportInputStream = this.getClass().getResourceAsStre
                file);
        if (reportInputStream == null) {
            reportInputStream = Thread.currentThread().getContextCla
                    .getResourceAsStream(file);
        }
        if (reportInputStream == null) {
            throw new IllegalArgumentException("Invalid report file:
        }

        return reportInputStream;
    }

    private static class IParameterDefnAdapter implements ParameterDefinitio
        private IParameterDefn parameterDefn;

        public IParameterDefnAdapter(IParameterDefn parameterDefn) {
```

```
                super();
                this.parameterDefn = parameterDefn;
        }

        @Override
        public String getName() {
                return this.parameterDefn.getName();
        }

        @Override
        public Class<?> getType() {
                switch (this.parameterDefn.getDataType()) {
                case IParameterDefn.TYPE_BOOLEAN:
                        return Boolean.class;
                case IParameterDefn.TYPE_DATE:
                case IParameterDefn.TYPE_TIME:
                case IParameterDefn.TYPE_DATE_TIME:
                        return Date.class;
                case IParameterDefn.TYPE_DECIMAL:
                        return Double.class;
                case IParameterDefn.TYPE_FLOAT:
                        return Float.class;
                case IParameterDefn.TYPE_STRING:
                        return String.class;
                default:
                        throw new IllegalStateException("Invalid parameter
                                        + this.parameterDefn.getDataType
                }
        }
    }
}
```

It extracts all the report information from BIRT file.

20.2 CRUD integration

CRUD integration can be done using classes from ar.com.oxen.nibiru.report.crud project. It provides:

- A CrudManager for reports:

```
package ar.com.oxen.nibiru.report.crud;

import java.util.ArrayList;
import java.util.LinkedList;
```

```
import java.util.List;

import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.crud.manager.api.CrudField;
import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.report.api.Report;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointManager;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionTracker;
import ar.com.oxen.nibiru.crud.utils.SimpleCrudField;

public class ReportCrudManager implements CrudManager<Report> {
        private List<Report> reports = new LinkedList<Report>();

        public ReportCrudManager(ExtensionPointManager extensionPointManager
                super();
                extensionPointManager.registerTracker(new ExtensionTracker<F
                        @Override
                        public void onRegister(Report extension) {
                                reports.add(extension);
                        }

                        @Override
                        public void onUnregister(Report extension) {
                                reports.remove(extension);
                        }
                }, Report.EXTENSION_POINT_NAME, Report.class);
        }

        @Override
        public String getEntityTypeName() {
                return Report.class.getName();
        }

        @Override
        public List<CrudField> getListFields() {
                List<CrudField> listFields = new ArrayList<CrudField>(1);
                listFields.add(new SimpleCrudField(ReportCrudEntity.REPORT_N
                                String.class, new SimpleCrudField.SimpleList
                return listFields;
        }

        @Override
        public List<CrudEntity<Report>> findAll() {
                return this.toEntity(this.reports);
        }
```

```
        @Override
        public CrudEntity<Report> findById(Object id) {
                return new ReportCrudEntity((Report) id);
        }

        @Override
        public List<CrudEntity<Report>> findByfield(String field, Object val
                // TODO Filtrar la lista por los campos
                return this.toEntity(this.reports);
        }

        private List<CrudEntity<Report>> toEntity(List<Report> reports) {
                List<CrudEntity<Report>> entities = new ArrayList<CrudEntity
                                reports.size());
                for (Report report : reports) {
                        entities.add(new ReportCrudEntity(report));
                }
                return entities;
        }
}
```

- A CrudEntity that wraps a report:

```
package ar.com.oxen.nibiru.report.crud;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.crud.manager.api.CrudField;
import static ar.com.oxen.nibiru.crud.manager.api.CrudField.FormInfo.GENERAL
import ar.com.oxen.nibiru.crud.manager.api.WidgetType;
import ar.com.oxen.nibiru.crud.utils.SimpleCrudField;
import ar.com.oxen.nibiru.report.api.Report;
import ar.com.oxen.nibiru.report.api.Report.ParameterDefinition;

class ReportCrudEntity implements CrudEntity<Report> {
        final static String REPORT_NAME_FIELD = "reportName";
        final static String REPORT_FORMAT_FIELD = "reportFormat";
        private Report report;
        private String format;
        private Map<String, Object> parameters;

        public ReportCrudEntity(Report report) {
```

```
        super();
        this.report = report;
        this.parameters = new HashMap<String, Object>();
}

@Override
public Object getId() {
        return report;
}

@Override
public List<CrudField> getFormFields() {
        List<CrudField> formFields = new LinkedList<CrudField>();

        /* Format field */
        formFields.add(new SimpleCrudField(REPORT_FORMAT_FIELD, Stri
                        null, new SimpleCrudField.SimpleFormInfo(Wi
                                        false, 0, GENERAL_TAB)));

        /* Parameter fields */
        for (Report.ParameterDefinition paramDef : this.report
                        .getParameterDefinitions()) {
                formFields.add(new ParameterDefinitionAdapter(paramD
        }

        return formFields;
}

@Override
public Object getValue(CrudField field) {
        return this.getValue(field.getName());
}

@Override
public Object getValue(String fieldName) {
        if (REPORT_NAME_FIELD.equals(fieldName)) {
                return this.report.getName();
        } else if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                return this.format;
        } else {
                return this.parameters.get(fieldName);
        }
}

@Override
public void setValue(CrudField field, Object value) {
```

```
        this.setValue(field.getName(), value);
}

@Override
public void setValue(String fieldName, Object value) {
        if (REPORT_NAME_FIELD.equals(fieldName)) {
                throw new IllegalArgumentException("Report name can'
                                + fieldName);
        } else if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                this.format = (String) value;
        } else {
                this.parameters.put(fieldName, value);
        }
}

@Override
public Report getEntity() {
        return report;
}

@Override
public String getEntityTypeName() {
        return Report.class.getName();
}

@Override
public Iterable<?> getAvailableValues(CrudField field) {
        return this.getAvailableValues(field.getName());
}

@Override
public Iterable<?> getAvailableValues(String fieldName) {
        if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                return this.report.getFormats();
        } else {
                throw new IllegalArgumentException(
                                "Field with no available values: " +
        }
}

private static class ParameterDefinitionAdapter implements CrudField
        private Report.ParameterDefinition paramDef;

        public ParameterDefinitionAdapter(ParameterDefinition paramD
                super();
                this.paramDef = paramDef;
```

```
                }

                @Override
                public String getName() {
                        return this.paramDef.getName();
                }

                @Override
                public Class<?> getType() {
                        return this.paramDef.getType();
                }

                @Override
                public ListInfo getListInfo() {
                        return null;
                }

                @Override
                public FormInfo getFormInfo() {
                        return new SimpleCrudField.SimpleFormInfo(WidgetType
                                        false, 9999, GENERAL_TAB);
                }
        }
}
```

- A CrudActionExtension which allows opening and runing a report:

```
package ar.com.oxen.nibiru.report.crud;

import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.crud.manager.api.CrudAction;
import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.crud.utils.AbstractCrudActionExtension;
import ar.com.oxen.nibiru.crud.utils.SimpleCrudAction;
import ar.com.oxen.nibiru.report.api.Report;

public class ReportCrudActionExtension extends
                AbstractCrudActionExtension<Report> {
        private final static String OPEN_REPORT = "openReport";
        private final static String RUN_REPORT = "runReport";
```

```
        private EventBus eventBus;

        public ReportCrudActionExtension(EventBus eventBus) {
                super(null);
                this.eventBus = eventBus;
        }

        @Override
        public List<CrudAction> getEntityActions(CrudEntity<Report> entity)
                List<CrudAction> actions = new ArrayList<CrudAction>(2);
                actions.add(new SimpleCrudAction(OPEN_REPORT, true, false, t
                                false, null));
                actions.add(new SimpleCrudAction(RUN_REPORT, true, true, fal
                                false, null));
                return actions;
        }

        @Override
        public CrudEntity<?> performEntityAction(CrudAction action,
                        CrudEntity<Report> entity) {
                if (action.getName().equals(OPEN_REPORT)) {
                        return entity;
                } else if (action.getName().equals(RUN_REPORT)) {
                        String format = (String) entity
                                        .getValue(ReportCrudEntity.REPORT_FO

                        Map<String, Object> parameters = new HashMap<String,

                        for (Report.ParameterDefinition parameterDef : entit
                                        .getParameterDefinitions()) {
                                parameters.put(parameterDef.getName(),
                                                entity.getValue(parameterDef
                        }

                        InputStream data = entity.getEntity().render(format,

                        this.eventBus.fireEvent(new ReportExecutedEvent(enti
                                        format, data));

                        return null;
                } else {
                        throw new IllegalArgumentException("Invalid action:
                }
        }
}
```

20.3 Report module

The ar.com.oxen.nibiru.report.module bundle provides a generic report module, which uses CRUD for providing report execution user interface.

It also provides a generic view which shows the executed report.

21 Workflow

TODO: Definir este módulo.

22 Mail

The ar.com.oxen.nibiru.mail.api bundle provides the API for sending e-mails. An e-mail is represented using the MailMessage class:

```
package ar.com.oxen.nibiru.mail.api;

import java.util.Collection;
import java.util.HashSet;

/**
 * A mail message.
 */
public class MailMessage {
        private String from;
        private Collection<String> to;
        private String subject;
        private String body;
        private String contentType;

        public MailMessage(String from, String subject, String body) {
                this(from, subject, body, "text/html");
        }

        public MailMessage(String from, String subject, String body,
                        String contentType) {
                super();
                this.from = from;
                this.subject = subject;
                this.body = body;
                this.contentType = contentType;
                this.to = new HashSet<String>();
        }
```

```
        public String getFrom() {
                return from;
        }

        public Collection<String> getTo() {
                return to;
        }

        public String getSubject() {
                return subject;
        }

        public String getBody() {
                return body;
        }

        public String getContentType() {
                return contentType;
        }

}
```

These messages can be sent using a MailService instance:

```
package ar.com.oxen.nibiru.mail.api;

/**
 * A service for sending e-mails.
 */
public interface MailService {
        /**
         * Sends a mail message.
         *
         * @param message
         *              The mail message
         */
        void sendMail(MailMessage message);
}
```

22.1 JavaMail implementation

The ar.com.oxen.nibiru.mail.javamail project provides a JavaMail based MailService implementation.

```
package ar.com.oxen.nibiru.mail.javamail;

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

import ar.com.oxen.nibiru.mail.api.MailException;
import ar.com.oxen.nibiru.mail.api.MailMessage;
import ar.com.oxen.nibiru.mail.api.MailService;

public class JavaMailService implements MailService {
        private Session session;
        private String username;
        private String password;
        private String host;
        private int port;
        private boolean startTls;

        public void init() {
                final Properties mailProperties = new Properties();

                if (username != null) {
                        mailProperties.put("mail.smtp.auth", "true");
                }
                if (startTls) {
                        mailProperties.put("mail.smtp.starttls.enable", "true");
                }
                mailProperties.put("mail.smtp.host", this.host);
                mailProperties.put("mail.smtp.port", String.valueOf(this.port));

                this.session = Session.getDefaultInstance(mailProperties,
                                new Authenticator() {
                                        protected PasswordAuthentication getPass
                                                return new PasswordAuthenticatio
                                        }
                                });
        }
```

```
@Override
public void sendMail(MailMessage message) {
        try {
                MimeMessage mimeMessage = new MimeMessage(this.session);
                mimeMessage.setFrom(new InternetAddress(message.getFrom(
                for (String recipient : message.getTo()) {
                        mimeMessage.addRecipient(Message.RecipientType.T
                                        new InternetAddress(recipient));
                }
                mimeMessage.setSubject(message.getSubject());

                if (message.getContentType() == null
                                | message.getContentType().equals("text/
                        mimeMessage.setText(message.getBody());
                } else {
                        mimeMessage.setContent(message.getBody(),
                                        message.getContentType());
                }

                Transport.send(mimeMessage);

        } catch (AddressException e) {
                throw new MailException(e);
        } catch (MessagingException e) {
                throw new MailException(e);
        }
}

public void setUsername(String username) {
        this.username = username;
}

public void setPassword(String password) {
        this.password = password;
}

public void setHost(String host) {
        this.host = host;
}

public void setPort(int port) {
        this.port = port;
}

public void setStartTls(boolean startTls) {
        this.startTls = startTls;
```

```
        }
}
```

23 Licensing

This module provides product license management. It relies on Oxen Java Commons license module. However, Oxen Java Commons classes just provide components for license request, authorization and validation. Extra functionality, such as storing and UI for requesting licences is required in order to integrate it with Nibiru. Such functionality is provided by this module and it is explained in the following sections.

23.1 License store

Once a license is received, it should be saved into a persistent store. This is what the ar.com.oxen.nibiru.license.store.api module provides.

The interface for storing and retreiving licenses is LicenseStoreManager:

```
package ar.com.oxen.nibiru.license.store.api;

/**
 * Manage for loading and saving licenses.
 */
public interface LicenseStoreManager {
        String GENERIC_MODULE = "genericModule";

        /**
         * Loads a license for a given module.
         *
         * @param module
         *              The module
         * @return A String representing the license
         */
        String loadLicense(String module);

        /**
         * Saves a license for a given module.
         *
         * @param module
         *              The module
         * @param license
         *              A String representing the license
         */
```

```
        void saveLicense(String module, String license);
}
```

The ar.com.oxen.nibiru.license.store.jpa project contains a JPA-based implementation for this API.

23.2 License module

The ar.com.oxen.nibiru.license.module project provides the UI for requesting a license. You can interact with this module by using the following event:

```
package ar.com.oxen.nibiru.license.module.event;

/**
 * Event for requesting a license.
 */
public class LicenseRequestEvent {
        private boolean showInvalidLicenseMessage;
        private Object callbackEvent;
        private String callbackTopic;

        /**
         * @param showInvalidLicenseMessage
         *              True if an invalid license message must be shown
         * @param callbackEvent
         *              The event to be fired once the license is loaded
         * @param callbackTopic
         *              The same, for topic
         */
        public LicenseRequestEvent(boolean showInvalidLicenseMessage,
                        Object callbackEvent, String callbackTopic) {
                super();
                this.showInvalidLicenseMessage = showInvalidLicenseMessage;
                this.callbackEvent = callbackEvent;
                this.callbackTopic = callbackTopic;
        }

        public boolean getShowInvalidLicenseMessage() {
                return showInvalidLicenseMessage;
        }

        public Object getCallbackEvent() {
                return callbackEvent;
        }

        public String getCallbackTopic() {
```

```
                return callbackTopic;
        }
}
```

Typically, you'll check the license using Oxen Java Commons classes (after retrieving it using the License store API). If no valid license is found, you can fire this event in order to display a window requesting a valid license. After a license is entered, the event/topic specified as callback are fired, so the licensed functionality can be executed again.

This module provides both presenters and views for ar.com.oxen.commons.license.impl.DefaultLicenseInfo. However, if a custom license information, tailor made presenter and views must be created.

TODO: Crear un proyecto service, para que se puedan personalizar los servicios de vista y presenter

23.3 Command line interface

In order to allow license authorization, a command line tool is provided. It is included in the ar.com.oxen.nibiru.license.cli module.

As before, it is based on ar.com.oxen.commons.license.impl.DefaultLicenseInfo license information. Also, and it identifies hardware using the MAC address. For a different license information and/or hardware identification, a custom authorizer must be created.

Part IV

Deployment

One of the advantages when developing under Nibiru framework is that your application can be deployed on both, OSGi and non-OSGi environments.

24 OSGi deployment

In order to deploy under an OSGi environment, you should break your application into OSGi bundles. However, there is some specific bundles and fragments that you may need to implement. They are explained in the following sections.

24.1 Webapp project

At least one webapp project must be created in order to publish a web application. However, if your applications is divided into many modules, usually they will share the same webapp.

The current implementation uses Vaddin. Because of this, you should create a Blueprint configuration like this one:

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
                http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/x
        ">

        <reference id="applicationAccessor"
                interface="ar.com.oxen.nibiru.ui.vaadin.api.ApplicationAccessor"

        <service interface="javax.servlet.Filter">
                <service-properties>
                        <entry key="urlPatterns" value="/*" />
                </service-properties>
                <bean class="ar.com.oxen.nibiru.http.utils.SessionHolderFilter"
        </service>

        <service ref="nibiruServlet" interface="javax.servlet.http.HttpServlet">
                <service-properties>
                        <entry key="alias" value="/sample" />
                        <entry key="widgetset" value="ar.com.oxen.nibiru.ui.vaad
                </service-properties>
        </service>

        <service ref="nibiruServlet" interface="javax.servlet.http.HttpServlet">
                <service-properties>
                        <entry key="alias" value="/VAADIN" />
                        <entry key="widgetset" value="ar.com.oxen.nibiru.ui.vaad
                </service-properties>
        </service>

        <bean id="nibiruServlet"
                class="ar.com.oxen.nibiru.ui.vaadin.servlet.NibiruApplicationSer
                <property name="applicationAccessor" ref="applicationAccessor" />
        </bean>

</blueprint>
```

25 Non-OSGi deployment

Support for non-OSGi environments is provided through ar.com.oxen.nibiru.standalone project. Such project contains the required dependencies for running the framework, pretty much as it is done in the ar.com.oxen.nibiru.targetplatform proyect. However, it is intended to be used in an standard Java environment, such as a servlet container.

The project also provides Spring configuration files for all the Nibiru modules. The ar/com/oxen/nibiru/standalone/context.xml file consolidates configuration files for a typical application.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springfran
        ">



        <import resource="classpath:/ar/com/oxen/nibiru/application/generic/conf
        <import resource="classpath:/ar/com/oxen/nibiru/application/generic/pres
        <import resource="classpath:/ar/com/oxen/nibiru/application/generic/view

        <import resource="classpath:/ar/com/oxen/nibiru/conversation/generic/con

        <import resource="classpath:/ar/com/oxen/nibiru/crud/ui/generic/presente
        <import resource="classpath:/ar/com/oxen/nibiru/crud/ui/generic/view/con

        <import resource="classpath:/ar/com/oxen/nibiru/extensionpoint/generic/c

        <import resource="classpath:/ar/com/oxen/nibiru/i18n/generic/conf/spring
        <import resource="classpath:/ar/com/oxen/nibiru/i18n/session/conf/spring

        <import resource="classpath:/ar/com/oxen/nibiru/session/http/conf/spring

        <import resource="classpath:/ar/com/oxen/nibiru/transaction/jta/conf/spr

        <import resource="classpath:/ar/com/oxen/nibiru/ui/vaadin/conf/spring/co

        <import resource="classpath:/ar/com/oxen/nibiru/commons/conf/spring/cont
        <import resource="classpath:/ar/com/oxen/nibiru/datasource/dbcp/conf/spr
        <import resource="classpath:/ar/com/oxen/nibiru/jpa/conf/spring/context.

</beans>
```

However, not all the modules are included in this file. Check the ar.com.oxen.nibiru.standalone.conf.spring package for available configurations.

Part V

License

The framework is distributed under Apache 2.0 license.