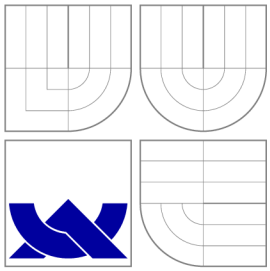
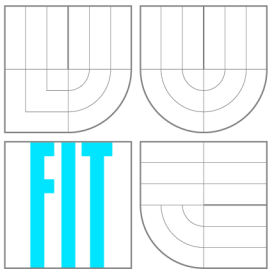


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PORTLETOVÁ IMPLEMENTACE HRY SUDOKU

SUDOKU GAME IMPLEMENTATION AS PORTLET

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ FIBICH
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Práce je zaměřena na problematiku vývoje komponent portálů, zvaných portlety. Stručně seznamuje s portály, zejména s jejich prostředím, vlastnostmi a schopnostmi. Hlavní část práce je věnována popisu standardů pro vývoj portletů. Na základě vysvětlených principů a s použitím dalších technologií je ukázán vývoj portletu, realizujícího hru Sudoku. Portlet byl vyvíjen pro firmu Red Hat, jakožto ukázkový portlet pro GateIn portál.

Abstract

The thesis is focused on the problem of the development of portal components, called portlets. It briefly introduces portals, especially their environment, characteristics and abilities. The main part describes standards for the development of portlets. Based on explained principles and with the usage of other technologies, a development process of a portlet, representing the Sudoku game, is shown. The portlet was developed for Red Hat company, as an exemplary portlet for GateIn portal.

Klíčová slova

Portál, Portlet, Sudoku, JSR 168, JSR 286, GateIn, REST, AJAX

Keywords

Portal, Portlet, Sudoku, JSR 168, JSR 286, GateIn, REST, AJAX

Citace

Ondřej Fibich: Sudoku game implementation as portlet, bakalářská práce, Brno, FIT VUT v Brně, 2012

Sudoku game implementation as portlet

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. Další informace mi poskytl odborný konzultant z firmy Red Hat, pan Mgr. Michal Vančo. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Fibich
6. května 2012

Poděkování

Rád bych poděkoval panu Ing. Radku Kočímu Ph.D. za trpělivé vedení práce a odbornému konzultantovi panu Mgr. Michalu Vančovi za poskytnuté vědomosti a pomoc při tvorbě práce. V neposlední řadě bych chtěl poděkovat slečně Bc. Zuzaně Bočkové za pomoc při jazykové korektuře.

Contents

Chapter 1

Introduction

The World Wide Web (WWW) became an irreplaceable gateway for the access to many types of knowledge and services. Because of the fact that a huge amount of data and information, circulating through the WWW, became very hard to find, online catalogues and web search engines were created for dealing with it. The next problem was that even if the required information were found, they were commonly placed in many different locations which prevented an effective usage of them. Portals were an answer to this issue. A single portal allows the aggregation of contents and the integration of services from different locations and sources. Many institutions understood advantages of portals and they built them for their customers, employees, citizens, etc. At present, portals belong to the most visited components of the WWW.

At the beginning the development process of creating of portals was similar to the process, used during the creation of web pages. A portal was usually developed from scratch. Portal creators had to implement its whole functionality by themselves. This time-consuming process was replaced by reusable portal solutions. Each portal creator developed a portal solution that was used for multiple portals. Specific functionalities for each portal were inserted by using of small components, called portlets. This new approach of building of portals allowed to supply only a general portal solution and portlets with specific functionalities, requested by a customer. The final portal was created by the customer who built it by combining of portlets on portal pages. An issue of this solution was that the customer may only used portlets that were supported by the portal solution. The issue led to creating of standards that allowed a portlet and a portal solution of different creators to be mutually compatible. This capability started a new era of portals because developers of portlets became independent of creators of portal solutions.

The thesis contains information which are required for understanding of the concept of portals. The majority of the thesis is dedicated to the description of two standards for the development of portals and portlets. A usage of these standards is demonstrated in a portlet application, developed in the practical part of the thesis. The portlet application represents a portlet implementation of the Sudoku game which was created for Red Hat company. It may serve as an exemplary portlet for an open-source portal, called GateIn portal, which is partially developed by this company. The main purpose of the portlet application is to show the possible way of building of portlet applications according to standards with a usage of REST and AJAX technologies.

Chapter 2

Portal

A portal is a web based application. The main goal of the portal is an aggregation of information from different sources and hosts. The aggregation is an action during which multiple inputs are transformed to a single output. The aggregated content is accessed by users whose are identified over an authentication. The portal authentication process also contains a uniform authentication to aggregated sources and hosts. Except from the aggregation the portal aims to create a personalised environment for users. This is very important because each user has different requests to functionality [10].

Since portals started to be important components of the WWW, many institutions came up with their own portals with different sets of end-user groups. It caused that portals split into several types, according to their usage [23]. These types are not strictly defined. Typically, each institution whose business is related to portals defines its own collection of portal types. On the other hand almost all these portal-related institutions have concurred in one portal type, called enterprise portal.

2.1 Enterprise portal

An enterprise portal is the most common portal type which mediates information, services and tools of a company or an organization to its employees, customers, vendors, etc. Key features of the portal are a personalized access, a single source of enterprise information, faster business decisions and improved business performances. Furthermore enterprise portals are helpful in corporations with multiple locations. In this type of the corporation sharing of information between separated departments may become difficult. The enterprise portal unifies and provides sharing of information and the cooperation among departments. This fact makes the enterprise portal an irreplaceable business tool.

The enterprise portal should include [23]:

- a single point of access – all required information and services can be accessed from a single point
- unified search across all information sources – information sources are not only accessed but they are also categorized and searchable from the portal
- personalization – a user can define a form and a type of the content using individual requests and preferences
- application integration – a user can manage a multiple application by a single request

- collaboration – information are shared between co-workers
- system security – a portal has to be well protected according to a security policy
- scalability – an ability of the system to extend with a new or an updated functionality in a short time

Enterprise portals are separated into two solutions. **Vertical portals** are the first solution. They are specified strictly in some area of the enterprise interest and designed to support specific functions, processes and services. The vertical portal is usually used for a customer service, resource planning, a sales force automation or a chain management [5]. An example of the vertical portal is the portal Hokej.cz[1] which is focused on ice hockey.

Horizontal portals are the second solution. The horizontal portal covers a wide area of the interest and it provides the functionality for supporting of an information flow, business activities and corporation processes [5]. An exemplary horizontal portal is the portal Centrum.cz.[2]

2.2 Portal environment

A portal environment contains many various elements. Some of them can be seen in the Figure 2.2. The most significant elements are [23]:

2.2.1 Web server

A primary function of a web server is answering to client requests by delivering a web page. The client who is sometimes called a user agent is commonly a web browser. The web browser and the web server communicate over the HTTP[3] stateless protocol. In the portal environment the web server mostly passes requests to the application server.

2.2.2 Application server

An application server is a software framework, providing an environment for running of application programs. The environment includes a container model for applications, services for applications and development tools. Application servers are used very often as distributors of user requests. Requests are distributed according to their contents to other physical servers. This is very useful for preventing from a server overload [19]. The application server is usually connected to a distributed network and separated to three architectonic layers [21].

- **Front-end** is a presentation layer which is separated to two types. The first type is a web browser-based interface. Web browser-base interfaces use mostly a markup language. The most popular markup languages for this action are HTML[4] or XHTML.[5] The second type is a graphical user interface (GUI) which is used on workstations.
- **Middle-tie** contains business logic applications.

[1] http://www.hokej.cz
[2] http://www.centrum.cz
[3] The Hypertext Transfer Protocol
[4] HyperText Markup Language
[5] eXtensible HyperText Markup Language

- **Back-end** includes database and a transaction server.

Application servers can cooperate with web servers. A basic cooperation is shown in the Figure 2.1. The Figure contains three participants. The first participant is a user. The user accesses the web server with a request. The web server is only capable of processing the static content. If the user request leads to a dynamic content, the web server will have to forward it to an application server. The application server processes the request using an application and sends a response back to the web server. The web server sends the response with the processed content to the user. The cooperation can be extended by a database server which contains data for a dynamic content and it is controlled by the application server and applications. However servers are separated services, there is no need to run them on a separated physical server. A single physical server can contain web, database and the application server [4].

There are many implementations of application servers, such as JBoss AS, Apache Tomcat, etc. JBoss AS and Apache Tomcat are open-source cross-platform Java EE-based application servers that differ in the amount of additional provided functionalities and libraries. JBoss AS is more robust than Apache Tomcat in this aspect.

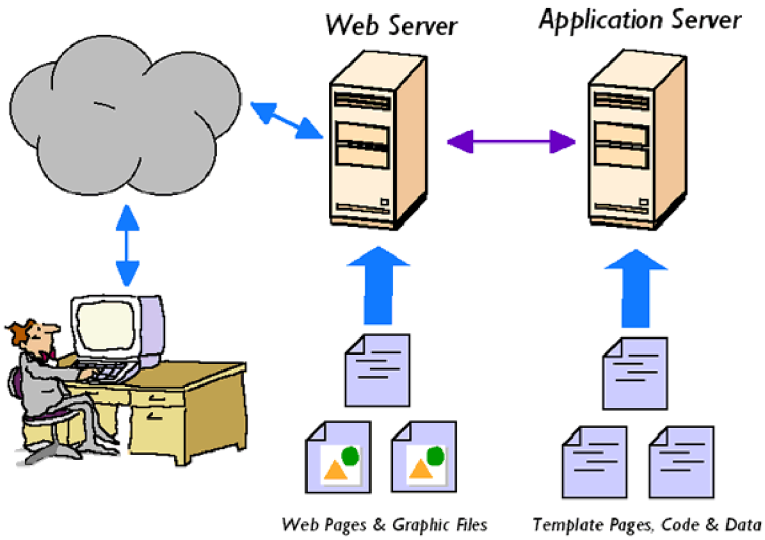


Figure 2.1: How an application server works [4]

2.2.3 Database

A database is not a required element of the portal environment but it is used very often. The database provides a storage of persistent data of the portal (e.g. user details, personalization settings, security options, etc.).

The most common type of the database is a relational database. Data in a relational database are organized into a set of related tables that are managed by SQL.[6] A representative of relational databases is for example HSQLDB.[7] In comparison with other database systems HSQLDB has a possibility to store a database in different kinds of storages (e.g. in a file, in the memory).

[6] Structured Query Language
[7] Hyper Structured Query Language Database

2.2.4 Gadgets and portlets

The presentation layer of the portal consists of gadgets and portlets. A gadget is a small application-specific component of the portal. On the other hand portlets are more robust components than gadgets [23]. This thesis is focused on portlets. More information about portlets can be found in the section 2.4.

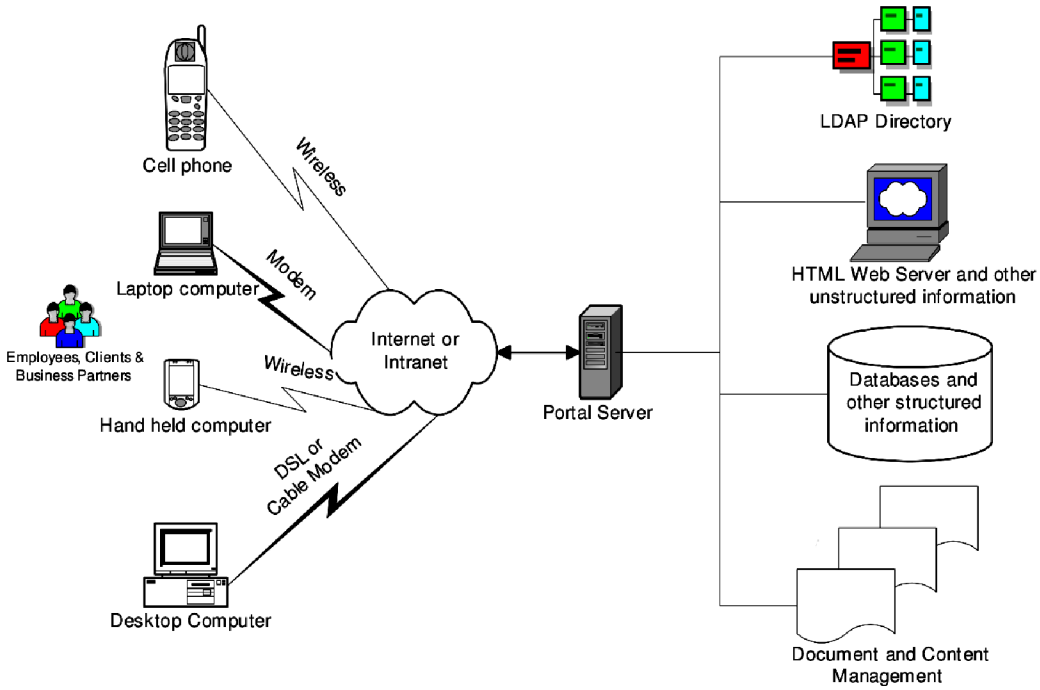


Figure 2.2: Generic portal ecosystem components [23]

2.3 GateIn portal

GateIn portal is an open source portal, released under the LGPL[8] licence. GateIn portal is a merge of two portals, JBoss Portal and eXo Portal, developed as a community project. Contributors of the portal are vendors of merged portals and community members. Main features of the GateIn portal are [14]:

- a capability of using portlets according to standards
- a skinnable, intuitive, multilingual user interface
- management of the content and the layout of each portal page
- user/group management and access permissions
- a dashboard functionality which enables to create a specific page for a user
- a collection of example portlets and gadgets

[8]The GNU Lesser General Public License

The JBoss Enterprise Portal Platform (EPP) is an enterprise edition of the GateIn portal with additional functionality and services such as the web content management functionality, legal protection and end-user support [13].

In spite of this section describes the GateIn portal, there are many other portal solutions. For example the most common used portal is the Liferay portal.[9] Another popular portal is the WebShere portal[10] which is developed by IBM. The description of the GateIn portal is placed in this thesis due to a fact that it was used as the deployment environment for the developed Sudoku portlet.

2.4 Portlet

A portlet is an application-specific pluggable user interface component of the portal. The application-specific property means that each portlet is created mostly for one purpose. A single portlet or a set of portlets creates a presentation layer of the portal [10]. Examples of the miscellaneous portlets can be found in the Figure 2.3.

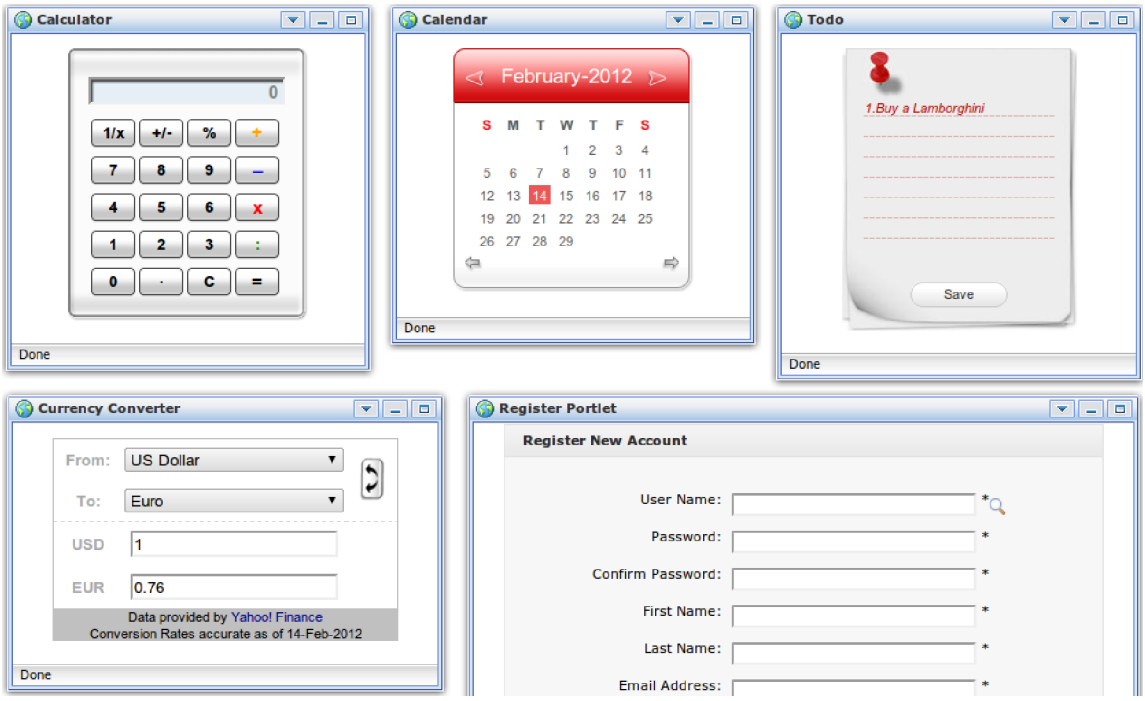


Figure 2.3: Examples of portlets

A single portal page is aggregated from generated contents of a set of the portlets. This is shown in the Figure 2.4. Moreover more instances of a portlet may be also displayed on a single portal page. The portlet generates a fragment of a document which commonly uses a markup language (e.g. HTML, XHTML).

An interaction between web clients and portlets is achieved via the request/response paradigm. Multiple portlets on a single page cause that a portlet cannot be requested directly from a web client. The communication is mediated through the portal [10].

[9] http://www.liferay.com
[10] http://www.ibm.com/software/websphere/

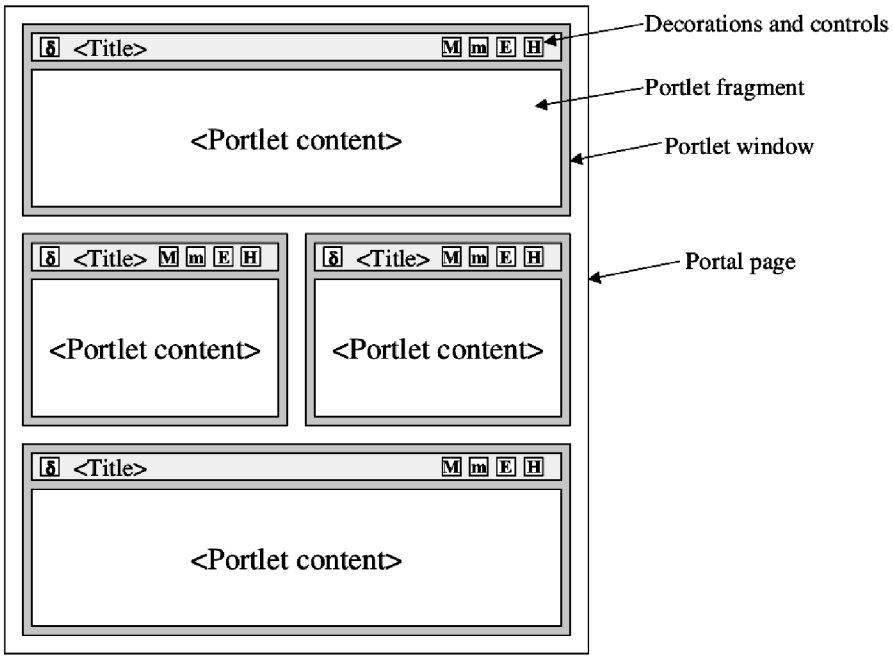


Figure 2.4: Elements of a portal page [11]

2.4.1 Portlet container

The portlet container is the runtime environment for portlets which manages the portlet lifetime. Moreover it receives requests from the portal and executes them on hosted portlets. The generated content of portlets is not aggregated by the portlet container which only forwards it to the portal. The content aggregation is handled by the portal, as shown in the Figure 2.5.

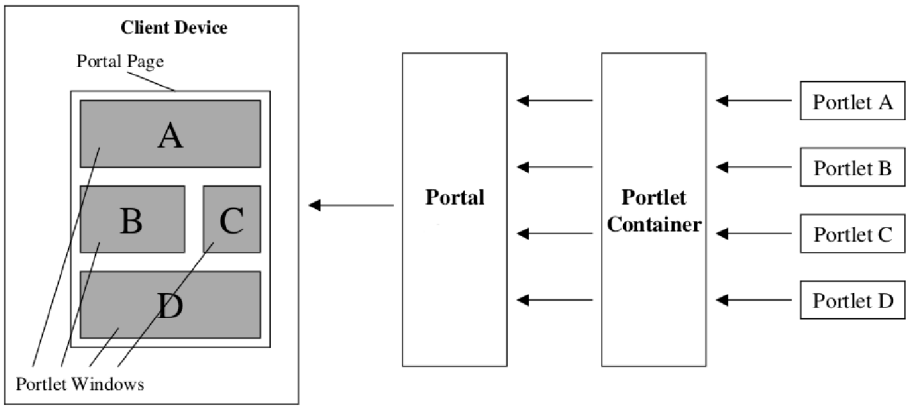


Figure 2.5: A cooperation between portlets, a portlet container, a portal and a client [11]

Chapter 3

Portlet development

Portals are mostly developed in **Java EE**[1] that is a part of Java platform designed and standardized for developing and deploying of enterprise applications and information systems [17]. The following text and chapters will not discuss other technologies (e.g. .NET,[2] PHP[3]) for creating of portals and portlets.

At the beginning portals were not designed to be compatible with each other. It meant that a portlet which was created for a portal could not be used in a different portal. This fact led to the creation of a standard specification for developing of portals and portlets. The specification guaranteed that a portlet, fulfilling the specification, could be run in a portal, designed according to the specification. Specifications were created by specialists from a group of corporations (e.g. IBM, Oracle and Sun Microsystems). A final version of the first portlet specification, called JSR 168, was released on 27 October, 2003. Following sections will describe the content of JSR 168 [11].

3.1 Specification JSR 168

The Portlet API is based on the Java EE specification. The proposal of the portlet specification was evolved from the servlet specification. A servlet is also a web-base component, managed by a container that generates a dynamic content [7]. Portlets and servlets differ in following aspects.

- A portlet generates a fragment but a servlet generates whole document.
- Despite servlets are bound to a URL,[4] portlets are not. Portlets must interact with a web client through a portal, using URL rewriting functions.
- Moreover portlets distinguish more types of requests and they have predefined states and modes which indicate the portlet state and the current function. Portlets are also capable of persisting of a configuration, customization data and access to user profile information.

These differences lead to a decision that portlets have to be a new component which leverages as much functionality as possible from the servlet specification. The most valuable

[1] Java Enterprise Edition
[2] .NET Framework - http://www.microsoft.com/net
[3] Hypertext Preprocessor - http://www.php.net
[4] Uniform resource locator

leveraged property is a capability of calling of servlets and JSPs[5] in order to generate the content of the portlet. This capability will be described in detail in the section 3.1.7.

The Portlet API is included in the `javax.portlet` package. The main abstraction of the Portlet API is placed in the interface `Portlet`. There is the `GenericPortlet` class which implements the `Portlet` interface and provides a basic functionality for creating a new portlet. Developers should extend this class to implement their portlets.

A set of portlets, servlets and JSPs creates a portlet application. More information about the portlet application can be found in the section 3.1.9.

3.1.1 Portlet life cycle

A life cycle of a portlet contains loading, an instantiation, an initialization, request handling and the end of service.

Loading and instantiation

Loading and instantiation are performed during a start or a delay of a portlet by the portlet container. A portlet class is loaded by the portlet container, using the Java ClassLoader. The Java ClassLoader is a part of Java Runtime Environment (JRE) that is responsible for dynamic loading of Java classes from different storages into the Java Virtual Machine (JVM). After loading of a portlet class an instance is made.

Initialization

The portlet instance is initialized by the portlet container, calling the `init` method of the `Portlet` interface. The method has one optional parameter: an instance of the class `PortletConfig` which contains a `PortletContext` object and configuration properties defined in the portlet descriptor. Configuration properties and the portlet descriptor will be described in the section 3.1.2. The `init` function may be overridden in a purpose of one-way action. Typically, one-way action is a connection to data sources and services (e.g. a database, a remote server and a file).

Request handling

The portlet may receive client requests after a proper initialization. Requests are passed to the portlet by the portlet container which manages the communication. There are two types of the request: an action request and a render request. The action request is processed by the `processAction` method of the `Portlet`. The render request is processed by the `render` method of the `Portlet`. Moreover the portlet container may pass the render request without any user's attempt to do it. Commonly, this situation occurs after passing an action request to the portlet. If there are more portlets on a portal page and a request is triggered to one of these portlets, a render request will be passed to all of these portlets after processing of the initial action request. This procedure is visualized in the Figure 3.1. Thus the portlet implementation must expect a render request at any time after the initialization.

The action request leads mostly to an update of a state of the portlet. The update is based on parameters of the request and performed by the `processAction` method. The `processAction` has two parameters: `ActionRequest` and `ActionResponse` objects. The

[5] JavaServer Pages

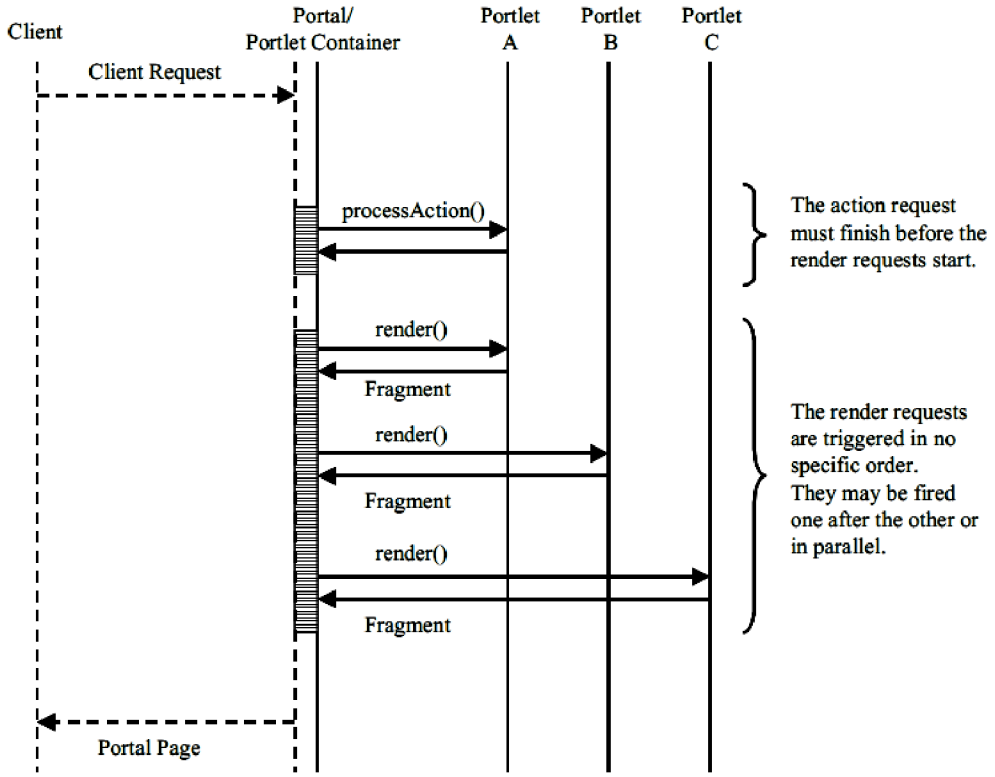


Figure 3.1: Handling of an action request [11]

`ActionRequest` object provides access to parameters and properties of the action request (`getProperty`, `getProperties`, `getPropertyNames`, `getParameter` and `getParameterMap` methods), the portlet mode (the `getPortletMode` method), the window state (the `getWindowState` method), the portlet context (the `getPortletContext` method), the portlet session (the `getPortletSession` method) and the portlet preferences (the `getPreferences` method). The `ActionResponse` object enables to set a redirection to a specific URL (the `sendRedirect` method) and changing of the portlet mode (the `setPortletMode` method) and the window state (the `setWindowState` method).

A generation of the content is made during a render request. The content is based on the current state of the portlet. A render request is processed by a `render` method which takes two parameters: `RenderRequest` and `RenderResponse` objects. The `RenderRequest` provides access to same information as the `ActionRequest`, except for parameters of the action request. The `RenderResponse` interface can generate a content using a writer. The example of a usage of the writer is in the Listing 3.1. Another way how to generate the content is to delegate the generation of content to a servlet or a JSP page. This way is explained in the section 3.1.7. The `RenderResponse` object is capable of setting the portlet title and the content type of the response.

```
1 renderResponse.setContentType("text/html");
2 PrintWriter writer = renderResponse.getWriter();
3 writer.write("Hello world!");
4 writer.close();
```

Listing 3.1: An example of portlet rendering

The `GenericPortlet` class implements the `render` method of the `Portlet`. The implementation sets the title of the portlet from data provided by the portlet deployment description and it dispatches the render request by the portlet mode stored in the `RenderRequest` object parameter of the `render` method. The class provides empty methods `doView`, `doEdit` and `doHelp`. Developers may override these methods for implementing of the generation of the content for the mode according to the name of the overrided method. The possible body of these methods is shown in the Listing 3.1.

Both action and render requests may be triggered by a portlet URL. Portlet URLs solve the problem of addressing multiple portlets on a portal page. The portlet container assigns user requests according to a portal URL, used for the request to the portlet which created the URL. There are two types of the URL corresponding to request types as well. The `PortletURL` interface, according to the specification, specifies portlet URLs. An object of the `PortletURL` may be created by `createActionURL` or by `createRenderURL` methods of the `RenderResponse` interface with the type according to the name of the method. Parameters may be added to a created URL by `setParameter` and `setParameters` methods of the `PortletURL`. The `PortletURL` also enables to specify a state and a mode of the portlet after processing of the request of the URL. Methods for this actions are the `setWindowState` and the `setPortletMode`. A secure connection over HTTPS[6] protocol for the request may be established by using the `setSecure` method. An example of creating and using of an action portlet URL is shown in the Listing 3.4.

End of service

The end of portlet services is determined by the portlet container. Before the destruction the portlet container calls the `destroy` method of the `Portlet` interface. The `destroy` method may be used for releasing of used resources and for saving of persistent states of the portlet.

3.1.2 Portlet configuration

A portlet configuration is provided for the `init` method of the `Portlet` interface by a `PortletConfig` object. The `PortletConfig` object contains initialization parameters, access to the portlet context and the resource bundle.

The portlet configuration is stored in the portlet deployment descriptor. The portlet deployment descriptor is an XML[7] document which includes the definition of all portlets (supported modes, title info, portlet class path, etc.) in the portlet application. More details about the portlet deployment descriptor are mentioned in the section 3.1.9.

Initialization parameters

An enumeration of names of initialization parameters is accessed by the `getParameterNames` method of the `PortletConfig` class. A single parameter value is returned by the `getParameter` method with its name as a parameter.

[6] Hypertext Transfer Protocol Secure
[7] Extensible Markup Language

Portlet resource bundle

A portlet resource bundle contains title-bar information resources and information for the categorization of the portlet stored in the portlet deployment descriptor or in a properties file whose path is defined by the `resource-bundle` element in the portlet deployment descriptor. The portlet specification defines three elements of the resource. Defined elements for portlet window title are the `title` and the `short-title`. An element `keywords` includes keywords for the categorization, separated by commas. The bundle resource is accessed by the `getResourceBundle` method of the `GenericPortlet` object.

Portlet context

A portlet context defines a portlet view of the portlet container and it is used for accessing to resources in the portlet application. The portlet specification defines the `PortletContext` interface which gains the access to the portlet context. An example of a usage of the `PortletContext` is shown in the section 3.1.7.

3.1.3 Portlet modes

A portlet mode is an indicator of the current performed portlet function. Each mode specifies a different task which generates a various content. The specification supports `VIEW`, `EDIT` and `HELP` modes. The support of the `VIEW` mode is required whereas others are optional.

- The `VIEW` **portlet mode** should generate a content which reflects the current state of the portlet. The mode can be implemented by an overriding `doView` method of the `GenericPortlet` class.
- The `EDIT` **portlet mode** should enable to change customization settings of the portlet to the user. The mode can be implemented by an overriding `doEdit` method of the `GenericPortlet` class.
- The `HELP` **portlet mode** should help a user to understand the portlet functionality and its user interface. The mode can be implemented by an overriding `doHelp` method of the `GenericPortlet` class.

Another unspecified mode, called a custom mode, can be added by the portal. If the developer wants to use a custom mode, he/she will need to define its support to the portlet deployment descriptor.

3.1.4 Portlet states

Portlet states indicate how much space is available for the portlet on a portal page. The portlet specification defines three states, other states can be added by portal vendors. The portlet should reduce the amount of a content according to the current portlet state.

- The `NORMAL` **portlet state** indicates that a portlet shares the portal page with other portlets.
- The `MAXIMIZED` **portlet state** indicates that a portlet occupies almost the whole portal page. An amount of a given space on a page is defined by the portal. Typically

when a portlet is maximized, the portal page contains only a toolbar or a navigation bar of the portal and a maximized portlet.

- The `MINIMIZED` **portlet state** indicates that a portlet should either generate a very small amount of a content or none.

3.1.5 Portlet preferences

Portlet preferences are typically used for storing of a configuration of a user customization. Preferences are permanently stored by the portlet container. A preference consists of a key and a set of values. A key which uniquely identifies values is defined as a `String` object and values as an array of `String` objects. The portlet specification defines the `PortalPreferences` interface for managing of preferences. This interface provides methods for accessing to values (`getNames`, `getValue`, `getValues` and `getMap` methods), methods for setting and modifying of preferences (`setValue` and `setValues` methods), a method for removing of preferences (the `reset` method) and the `store` method which commits all changes and makes preferences persistent.

Portlet preferences can be only set and modified within the `processAction` method of the `Portlet`.

Default preferences can be defined in the portlet deployment descriptor. Preferences, defined in the descriptor, can be read-only. The read-only state is indicated by the `isReadOnly` method.

3.1.6 Sessions

Sessions allow to store information about users who request the portlet. Sessions are stored only for the current session. The `PortletSession` interface tracks user sessions and provide a functionality for manipulating with sessions.

- The `setAttribute` method sets or modifies a session attribute.
- The `getAttribute` method accesses a session attribute.
- The `removeAttribute` method removes a session attribute.
- The `getAttributeNames` method accesses an enumeration of all names of stored attributes.

A session attribute contains a set of name, value and scope. While a name which uniquely identifies the session attribute is defined as a `String` object, a value is defined as an `Object` object. A scope defines a visibility of the session attribute. Possible values of the scope are the `APPLICATION_SCOPE` and the `PORTLET_SCOPE`. The scope with a value `PORTLET_SCOPE` indicates that the session attribute can be only accessed from the portlet which creates the attribute. On the other hand the `APPLICATION_SCOPE` value of the scope enables access for all portlets from the same portlet application which is the only way how portlets can communicate with each other according to the JSR 186.

3.1.7 Dispatching requests to servlets and JSPs

The portlet fragment markup can be either generated out of the portlet in servlets or JSPs. For these purposes the `PortletContext` interface provides access to a `PortletRequestDispatcher` object. The object must be only accessed in the `render` method of the `Portlet`. There are two methods for returning of the object from the portlet context.

- The `getRequestDispatcher` obtains a `PortletRequestDispatcher` object for dispatching to a servlet or a JSP page on a path which is relative to the portlet context. The path is given as a parameter.
- The `getNamedDispatcher` also obtains a `PortletRequestDispatcher` object for dispatching to a servlet or a JSP page. The difference is that the parameter of the method is a name, not a path. The name is assigned to a path.

After obtaining of an object of the `PortletRequestDispatcher` interface, a method `include` may be called for including of a servlet or a JSP page. The `include` method takes two parameters, a `RenderRequest` object and a `RenderResponse` object. An example of a usage of the `PortletRequestDispatcher` interface is shown in the Listing 3.2.

```
1 PortletRequestDispatcher prd = context.getRequestDispatcher("/page.jsp");
2 prd.include(renderRequest, renderResponse);
```

Listing 3.2: An example of a usage of the `PortletRequestDispatcher` interface

Servlets which were included have an indirect access to the functionality of request and response objects of the portlet through `HttpServletRequest` and `HttpServletResponse` interfaces, defined by the servlet specification.

Portlet tag library

The portlet tag library provides a direct access of request and response objects and other functionalities (e.g. creating of portlet URLs) to a JSP page, included by the portlet. The usage of the library must be defined in a JSP page by adding of the first line of the Listing 3.3. For a direct access to `RenderRequest`, `RenderResponse` and `PortletConfig` objects the `<portlet:defineObjects/>` tag must be included in the JSP page.

```
1 <%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
2 <portlet:defineObjects/>
3 Hello, <%=renderRequest.getRemoteUser()%>!
```

Listing 3.3: An example of the JSP page with the portlet tag library

The portlet tag library defines following tags.

- The `namespace` tag produces a string which is unique for the current portlet. Typically, this unique value is used for the namespacing of HTML DOM[8] identificators or ECMAScript[9] global functions and variables. The namespacing is required for achieving of the proper functionality on a portal page with multiple instances of the same portlet.

[8] The Document Object Model
[9] ECMAScript is a language, commonly used for client-scripting in HTML documents

- The `param` tag defines a parameter of the `actionURL` or the `renderURL` tag. The tag provides `name` and `value` attributes.
- The `actionURL` tag creates a URL which leads to an action of the current portlet. Parameters may be added, using the `param` tag. The tag provides following optional attributes. The `windowState` and `portletMode` attributes define the portlet state and mode in which the portlet will be after an execution of the action. The secure connection can be established, using the `secure` attribute with the `true` value. The `var` tag indicates whether the URL is directly written to the JSP page or is stored in the variable with a name, defined in a value of the tag. An empty value indicates the directly written URL. An example of the tag is shown in the Listing 3.4.
- The `renderURL` tag works as the `actionURL` tag only with the exception: the `actionURL` leads to an action and the `renderURL` tag leads to performing of a portlet render request.

```
1 <portlet:actionURL portletMode="view" var="url_switch">
2     <portlet:param name="function" value="switchToViewMode" />
3 </portlet:actionURL>
4 <a href="<%=url_switch%>">Switch to the VIEW mode</a>
```

Listing 3.4: An example of the `actionURL` tag

3.1.8 Security

A security of the portlet can be achieved on the programmatic level by using of following methods from the `Request` interface and a definition of user roles. The programmatic level of the security ensures a restriction of access to the portlet by a user.

- The `getRemoteUser` method returns an identificator, used for the authentication of a logged user. If a user is not logged to the portal, the `null` will be returned.
- The `isUserInRole` method checks if a user is in a role, given by a parameter of the method.
- The `getUserPrincipal` method returns a principal name of a logged user in a form of a `java.security.Principal` object.

Roles can be defined by the portal and mapped to portlets using `security-role` tag of the portlet deployment descriptor. A definition of roles is not a part of the specification.

Another possible security option is a usage of the secured communication over SSL.[10] It can be achieved by the `setSecure` method of a `PortletURL` object and by propper setting of the deployment environment of the portal.

3.1.9 Deployment

An action which delivers a software system to a user is called a deployment. A deployment of the portlet application is directed by a configuration, stored in deployment descriptors.

[10] Secure Sockets Layer

The portlet application is deployed in a form of one application archive (WAR[11] file). The application archive contains all resources, portlets, servlets and deployment descriptors which are required for independent running of the portlet application.

The portlet application has two deployment descriptors. The first deployment descriptor specifies web application resources. The web deployment descriptor is stored in the application archive in a form of an XML document, located at the path `/WEB-INF/web.xml`. The web deployment descriptor also contains important elements for the portlet application, such as a description of the application (the `description` tag), a portlet application name (the `display-name` tag) and an element for security role mapping (the `security-role` tag).

The second deployment descriptor is only related to a configuration of the portlet. Moreover it is an XML document, stored in the application archive and located at the path `/WEB-INF/portal.xml`. Portlet deployment descriptors contain definitions of the portlet application and its portlets. An example of the portlet deployment descriptor is shown in the Figure 3.6.

The application archive may also provide information about a version of the application. It is placed in the manifest file at the path `/META-INF/MANIFEST.MF` in the application archive. An example of the manifest file is shown in the Listing 3.5.

```
1 Manifest-Version: 1.0
2 Built-By: Ondrej Fibich
3 Built-Jdk: 1.6.0_26
4 Implementation-Title: Sudoku_Game
5 Implementation-Version: 1.0.0
```

Listing 3.5: An example of the `MANIFEST.MF` file

3.2 Specification JSR 286

After the first version of the Portlet API, introduced in JSR 168, became popular with portal vendors and developers, new and postponed requests for functionalities were defined. The most important request was a capability of building of integrated composite applications, out of portlets. This capability was not defined by the first specification. Requests led to a creation of the second portlet specification. A final version of the second specification, called JSR 286, was released on the 12th of July in 2008. This version is the latest revision of the portlet specification.

The JSR 286 extends the first specification while maintaining the backward compatibility. The backward compatibility was an important issue because there were many portals and portlets, developed according to the first specification. New main features of the specification are a coordination between portlets through events and public render parameters, resource serving, portlet filters and a support of the AJAX[12] technology [12]. The following chapters will describe news in the JSR 286 [10].

3.2.1 Coordination between portlets

The coordination between portlets, according to the JSR 168, could be achieved by using Sessions with limitations. The most important limitation was that the coordination was

[11] Web application ARchive
[12] Asynchronous JavaScript and XML

restricted to work only between portlets in the same portlet application [11]. JSR 286 came up with two new types of coordination which do not have the limitation of their predecessor. These types of coordination allow a cooperation between portlets, developed by different developers. Both types use the loosely coupled publish/subscribe model that enables an independent development of coordinating portlets. A developer only defines data that are understandable by a portlet and a connection between portlets is made at runtime of the portlet application. Using coordinating portlets, portal administrators and users can build an integrated composite application by combining of these components without any intervention to the source code of portlet applications [12].

Portlet events

Portlet events react to actions or a change of a state that are typical results of user interactions. Events are generated by a portlet or by the portlet container. A portlet which receives an event may change its state or create new events that should be delivered to other portlets. Events are placed after action requests in the life cycle of portlets. Thus, received events can not directly change the content of a portlet.

The JSR 286 specification defines the `EventPortlet` interface that contains only one method, called `processEvent`. The `processEvent` method takes two parameters, an object of the `EventRequest` and an object of the `EventResponse`. Portlets must implement this interface in order to receive events. A predefined event may be triggered by the `setEvent` method of a response object, during processing of an action or an event. Events are identified by QNames[13] and may contain a value which must be a serializable object. An access to a name, a QName and a value are provided by methods `getName`, `getQName` and `getValue` of an `Event` object which is gained from the `getEvent` method of the `EventRequest`.

A configuration of portlet events is placed in the portlet deployment descriptor. A portlet event is defined by the `event-definition` tag. Relationships between portlets and events are divided into two situations which reflect that the defined event can be published or processed by a portlet. Publishing is defined by the `supported-publishing-event` tag. An event may be published by the `setEvent` method that is called inside the `processAction` method or the `processEvent` method of the portlet, as shown in the Listing 3.7. An element of the `supported-processing-event` tag defines a processing relationship. In this situation a portlet listens to incoming events and processes them in the `processEvent` method. An example of processing is shown in the Listing 3.8.

```
1  <portlet-app ...>
2      <portlet>
3          <portlet-name>PortletA</portlet-name>
4          <supported-publishing-event>
5              <qname xmlns:x="http://fit.vutbr.cz/portal/portlets/ns">x:event</qname>
6          </supported-publishing-event>
7      </portlet>
8      <portlet>
9          <portlet-name>PortletB</portlet-name>
10         <supported-processing-event>
11             <qname xmlns:x="http://fit.vutbr.cz/portal/portlets/ns">x:event</qname>
12         </supported-processing-event>
13     </portlet>
14     <event-definition>
```

[13] XML qualified names - http://www.w3.org/TR/xmlschema-2/#QName

```
        <qname xmlns:x="http://fit.vutbr.cz/portal/portlets/ns">x:event</qname>
        <value-type>org.gatein.examples.games.sudoku.entity.Service</value-type>
    </event-definition>
</portlet-app>
```

Listing 3.6: An example of the definition of an event

```
public void processAction(ActionRequest req, ActionResponse res) {
    QName qname = new QName("http://fit.vutbr.cz/portal/portlets/ns" , "event");
    res.setEvent(qname, new Service(null, "Rahan.wz.cz"));
}
```

Listing 3.7: An example of a `processAction` method of the PortletA from the Listing 3.6

```
public void processEvent(EventRequest req, EventResponse res) {
    Event e = req.getEvent();
    if (e.getName().equals("event"))
        res.setRenderParameter("serviceName", ((Service) e.getValue()).getName());
}
```

Listing 3.8: An example of a `processEvent` method of the PortletB from the Listing 3.6

Public render parameters

Public render parameters are properties of a portlet that are shared to other portlets. While portlet events create additional event calls, public renders parameters are commonly stored in the URL. Thus, public render parameters enable the browser navigation and bookmarking to the end-user.

A public render parameter must be declared in the portlet deployment descriptor by the `public-render-parameter` tag. Portlets that share a parameter may read, edit and delete it in any part of their life cycle. Sharing as well as the declaration must be defined in the portlet deployment descriptor by the `supported-public-render-parameter` tag. An example of the portlet deployment descriptor with public render parameters is shown in the Listing 3.9.

```
<portlet-app ...>
    <portlet>
       <portlet-name>PortletA</portlet-name>
       <supported-public-render-parameter>param</supported-public-render-parameter>
    </portlet>
    <portlet>
       <portlet-name>PortletB</portlet-name>
       <supported-public-render-parameter>param</supported-public-render-parameter>
    </portlet>
    <public-render-parameter>
       <identifier>param</identifier>
       <qname xmlns:x="http://fit.vutbr.cz/portal/portlets/ns">x:param</qname>
    </public-render-parameter>
</portlet-app>
```

Listing 3.9: An example of declaring and sharing public render parameters

Portlets which support a public render parameter may access it, using the `getPublicParameterMap` method of a request and edit or delete it, using `setRenderParameter` or `removePublicRenderParameter` methods of a response. Each public render parameter is identified by a QName.

3.2.2 Resource serving

Resources can be served by two different ways in the portlet application. The first way is used for static resources. It uses direct links with URLs which direct to a resource, stored in the portlet application. This type of resource serving is already supported in the JSR 168 and it is typically used for accessing images, scripts, etc. The second way was added to the JSR 286 in order to provide serving of dynamic resources by portlets. While direct links are created for static resources, links that direct back to the origin portlet are created for dynamic resources. This fact allows to serve resources which may be protected by the portlet security and can leverage the portlet context. The following text will describe a mechanism which is defined for serving of dynamic resources.

A resource may be served by a portlet by implementing the `ResourceServingPortlet` interface which has a single method, called `serveResource`. In the life cycle of the portlet the `serveResource` method follows the `render` method. A single portlet can serve several resources, each resource is determined by a resource identifier and by a HTTP method which invokes serving. HTTP methods, such as POST, PUT or DELETE are commonly used for changing a state of the portlet. On the other hand GET method should be used for getting the current state or some other data without changing them. The `serveResource` method takes two parameters. The HTTP method and a resource identifier may be accessed from the first parameter which is an object of the `ResourceRequest` interface. For these purposes the interface provides `getResourceID` and `getMethod` methods. The interface also provides a functionality for accessing the portlet mode, the window state, request information, etc. The second parameter is an object of the `ResourceResponse` interface whose main purpose is to create an output of a type which is defined by the `setContentType` method. The output is made by a writer that is available by the `getWriter` method of the response.

Request URLs which head to a served resource may be created by the `createResourceURL` method of the `PortletRequest` which is an ancestor of the `ActionRequest` and the `RenderRequest`. The `createResourceURL` method returns an `ResourceURL` object. Another way of creating a resource URL is to create it by the `portlet:resourceURL` JSP tag. An example of a usage of the tag is shown in the Listing 3.10.

```
1 <portlet:resourceURL var="helloURL" id="helloURL" escapeXml="false">
2     <portlet:param name="name" value="Mike" />
3 </portlet:resourceURL>
```

Listing 3.10: An example of the creation of a resource URL by a JSP tag

The request, invoked by a URL which is defined in the Listing 3.10, can be processed in the `serveResource` method, as shown in the Listing 3.11.

Dynamic resources are cached according to a caching level. Levels indicate which changes in the portal page should invoke reloading of cached resources. `getCachability` and `setCachability` methods of the `ResourceURL` allow managing of caching levels. There are three levels: `PAGE` for keeping the cached resource until any state of the page changes, `PORTLET` for keeping the cached resource until a portlet state changes and `FULL` which indicates that keeping of the cached resource does not depend on any state change. If a caching

level of the resource URL is not set, a level of a parent resource will be used. If there is no parent, the `PAGE` level will be used.

```
1 public void serveResource(ResourceRequest req, ResourceResponse res)
2         throws PortletException, IOException {
3     if ("helloURL".equals(req.getResourceID()))
4         res.getWriter().write("Hello " + req.getParameter("name"));
5 }
```

Listing 3.11: An example of the `serveResource` method

Serving fragments through portlets

The resource serving mechanism is often used for serving fragments through portlets. Typically, a content of a portlet is changed or updated asynchronously, according to a fragment which is obtained from the `serveResource` method by an AJAX request to a resource URL. Despite this method allows to change a content of the portlet, it has several restrictions (e.g. it is not allowed to change the portlet state).

3.2.3 Portlet Filter

Filters are Java components that are based on servlet filters. Typically, filters are used for pre-processing of requests, post-processing of responses or changing output data without an interference to the source code of the portlet. Filters are provided for all life cycle methods (`render`, `processAction`, `processEvent` and `serveResource`). The life cycle of a filter contains loading, the instantiation, the initialization, filtering and the destruction.

Interfaces and classes, related to filters, are located in the `javax.portlet.filter` package. There are four interfaces, one for each life cycle method of the portlet. Interfaces have the `doFilter` method which performs filtering. Moreover they extend the `PortletFilter` interface which provides the rest of life cycle methods of a filter, such as the `init` method for the initialization and the `destroy` method for the destruction. The `init` method takes an object of the `FilterConfig` that provides an access to the portlet context and to the configuration of the filter, given by the portlet deployment descriptor. Each `doFilter` method of a filter takes three parameters: a request object, a response object and a `FilterChain` object. Response and request object types are given by a name of the interface that defines the method (e.g. the `ActionRequest` for the `ActionFilter`). An object of the `FilterChain` may create a sequence of filters, associated with a single life cycle method of the portlet. When all filters in the sequence are processed (their `doFilter` methods are called), the filtered life cycle method of a portlet is triggered. The next filter in the chain is invoked by calling of the `doFilter` method of the `FilterChain`. The body of the `doFilter` method of a filter should be separated into two parts, as shown in the Listing 3.12. The source code, placed in the first part, is triggered before the life cycle method of the portlet and the second part is triggered after its processing. If the `doFilter` method of the `FilterChain` is not called in the method, the filtered life cycle method will not be processed.

```
public void doFilter(ActionRequest request, ActionResponse response,
        FilterChain chain) throws IOException, PortletException {
    // some code triggered before processing of the action request (e.g. wrapping)
    chain.doFilter(request, response);
    // some code triggered after processing of the action request (e.g. wrapping)
}
```

Listing 3.12: An example of a `doFilter` method

A filter interface (e.g. `ActionFilter`) should be implemented in order to create a filter. A single filter can be used for filtering of multiple life cycle methods by implementing multiple filter interfaces. Request and response objects (e.g. `ActionRequest`, `ActionResponse`) may be wrapped in the `doFilter` method, using predefined wrappers (e.g. `ActionRequestWrapper`, `ActionResponseWrapper`). The wrapping functionality is used for the modification of request or response object. Wrapped request or response object is passed to the `doFilter` method of a `FilterChain` object instead of the original object.

Filters are configurated by the portlet deployment descriptor. An element of the `filter` tag contains information about the filter, such as a name, a Java class, supported life cycle phases of filtered portlets and initialization parameters. An element of the `filter-mapping` tag defines an association between a filter and a portlet or a group of portlets. An example of a filter configuration is shown in the Listing 3.13.

```
<filter>
    <filter-name>ActionFilter1</filter-name>
    <filter-class>com.test.filter.ActionFilter1</filter-class>
    <lifecycle>ACTION_PHASE</lifecycle>
    <init-param><name>par1</name><value>val1</value></init-param>
</filter>
<filter-mapping>
    <filter-name>ActionFilter1</filter-name>
    <portlet-name>FilteredPortlet</portlet-name>
</filter-mapping>
```

Listing 3.13: An example of a portlet filter configuration

3.2.4 Caching

Content caching is used for improving of request time of the portal and reducing of the load on servers. The portlet specification provides an expiration based caching mechanism, separated to each portlet. A content of a portlet is cached until the expiration time passes or an action or a event request is triggered. An expiration time may be set in the portlet deployment descriptor, using the `expiration-cache` tag or during runtime, using the `setProperty` method of the `RenderRequest` interface with the first parameter which is set to a value of the `EXPIRATION_CACHE`. The behaviour of the caching mechanism may be also configured by a scope. There are two defined scopes. The `PRIVATE_SCOPE` is used for data which cannot be shared between users. On the other hand the `PUBLIC_SCOPE` scope is used for a shared content. The scope of the request as well as the expiration time may be specified in the portlet deployment descriptor or by the `setProperty` method with the `CACHE_SCOPE` key.

Caching of dynamic resources is not based on the expiration based mechanism. The description of resource caching is available in the section 3.2.2.

3.2.5 Annotations

The JSR 286 specification defines several annotations. These annotations were created in order to reduce the source code for dispatching of requests in life cycle methods of a portlet. An annotation is placed in front of the life cycle method which processes or renders the request, identified by a name or a QName. There are following annotations: `@ProcessAction(name="<name>")` for an action, `@ProcessEvent(qname="<qname>")` for an event and `@RenderMode(name="<name>")` for a render action. An example of a usage of the `@ProcessEvent` annotation is shown in the Listing 3.14.

```
1 @ProcessEvent(qname="{http://fit.vutbr.cz/portal/portlets/ns}event")
2 public void event1(EventRequest req, EventResponse res) {
3     Event e = req.getEvent();
4     res.setRenderParameter("serviceName", ((Service) e.getValue()).getName());
5 }
```

Listing 3.14: An example of the `@ProcessEvent` annotation that reduces the example in the Listing 3.8.

Chapter 4

Analysis and proposal

A practical task of this thesis was to develop a portlet implementation of the Sudoku game for Red Hat company. The final product should be a part of the GateIn portal and the JBoss Enterprise Portal Platform as an example portlet that shows how portlets may be designed and developed. The secondary purpose of the portlet is its usage in a process of the quality assurance of both portals.

4.1 Analysis

The Sudoku game is a logic-based number placement puzzle. There are many versions of the game. The basic version has 81 cells, placed in 9 horizontal rows and 9 vertical columns. Except for rows and columns, cells are divided to 9 uniformed boxes with a size of 3x3. Each cell can contain a number value in a range from 1 to 9. A solved game contains all values from the range in each row, column and 3x3 box. A Sudoku puzzle is created by omitting of few number values from a solved game. A difficulty of a puzzle is given by a count and a placement of omitted values [22].

4.1.1 Specification of requests

The portlet implementation of the Sudoku game should include:

- generating of new games on demand
- loading and playing of games from periodical remote services
- loading and playing of previously played or saved games of a user (games must be unfinished)
- loading and playing of games played by other users
- capabilities to check, to reset or to pause the current played game of a user
- visualizing of statistics about the current game of a user
- visualizing of information about the best solvers and summarized statistics about all games
- managing of periodical remote services by an administrator

- providing of a skinnable user interface

The portlet must be capable of operating for multiple users at single moment. Details of logged users are provided by the portal. An access to the portlet must be provided for a user even if he/she is not logged in the portal. In this situation the implementation must not allow following actions to an anonymous user: saving of the current game, loading of previously saved or played game solutions and changing of a skin of the user interface. An administrator as well as a user is distinguished, using information which are provided from the portal.

The portlet should comply with the second portlet specification (JSR 286) in order to achieve an easy portability to various types of deployment environments. The final product must be compatible with following portals and their deployment environments:

- GateIn 3.2.0.Final with JBoss AS 5.1.0
- GateIn 3.2.0.Final with JBoss AS 6.0.0
- GateIn 3.2.0.Final with Tomcat 6
- GateIn 3.2.0.Final with Tomcat 7
- JBoss Enterprise Portal Platform 5.2.0

4.1.2 Available solutions

Due to the fact that the Sudoku game is very popular, the portlet developed in this thesis was the first of its kind. There are some gadget implementations, provided by iGoogle.com[1] but most of these implementations are only fragments of a page from different web pages, inserted in a form of a gadget.

On the other hand there are many web services that provide and allow to play games. Typically, each service contains its own application for playing games. Some of these services were used as periodical remote services for obtaining new games. For example the DailySuDoku.com[2] server publishes Sudoku games with various difficulties every day.

4.1.3 Use cases

The behaviour of the application from an end-user point of view in a form of a use case diagram is shown in the Figure 4.1. The use case diagram is a part of UML[3] that is a graphical language, used for the representation, the specification, proposing and documenting of software systems.

The diagram contains three types of end-users: an administrator, a user and an anonymous user. Anonymous users can perform actions over games, such as loading, creating, playing, pausing, checking, resetting and accessing of statistics. Users share all actions of anonymous users and add personalized actions, such as loading and saving of game solutions, changing of a skin and loading of previously played unfinished game solutions. Administrators have capabilities of users, moreover they can manage periodical remote services.

[1] http://www.igoogle.com
[2] http://www.dailysudoku.com/
[3] Unified Modeling Language

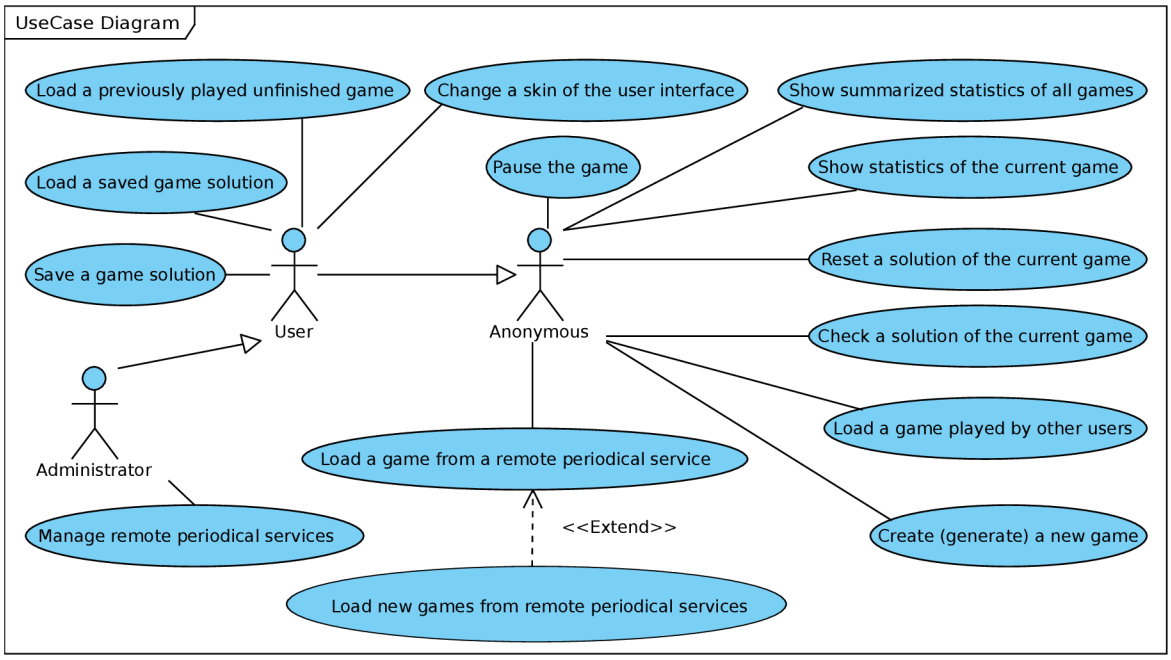


Figure 4.1: All use cases of the Sudoku portlet

4.2 Proposal

The application is separated in two individual parts. The first part, called "Client side", is responsible for providing of a logic for playing of games, maintaining of game states and accessing to statistics and game lists. The Client side communicates with the second part, called "Server side", in order to get, create or edit persistent data of games.

4.2.1 Server side

The Server side of the application is a persistent storage of services, games, solutions of games and saved game solutions with a set of operations over them. A scheme of stored data is shown in a form of a conceptual model in the Figure 4.2. The conceptual model contains five entities: games, game solutions, saved game solutions, last played game solutions and services. Information about users are provided by the portal. Thus, a user entity is not used.

- Basic immutable information about a Sudoku game are placed in the entity, called Game. The Game entity contains initialization values, such as the creation time and initialization field values that contain fixed values of a Sudoku game. A game type specifies how a game was created. There are two types: generated and service. A Game entity with the generated type was algorithmically generated within this application with a difficulty, specified by a difficulty attribute. On the other hand a Game entity with the service type was obtained from a periodical remote service. If the game is of the service type, it will contain a reference to a Service entity.
- In order to play a game, a Game solution entity must be created. Each solution is owned by a user who is distinguished by his/her identificator. The entity persists following attributes: a time stamp of the start, lasting in seconds, field values, a

counter of usages of the solution check, user's rating and an indicator which denotes whether the solution is solved or not.

- A game solution can be saved during solving, using the Saved game solution entity. The Saved game solution entity contains an image of the current state of solution, identified by a name.
- A last played game solution is an association between a portal user and one of his/her game solutions which is persisted in the Last played game solution entity.
- The Service entity encases information about a periodical remote service, such as a name, a URL, an expiration time and an indicator for enabling/disabling. The expiration time, stored in the check time attribute, contains a duration in seconds after which the service, located at the URL, should be checked for a new game.

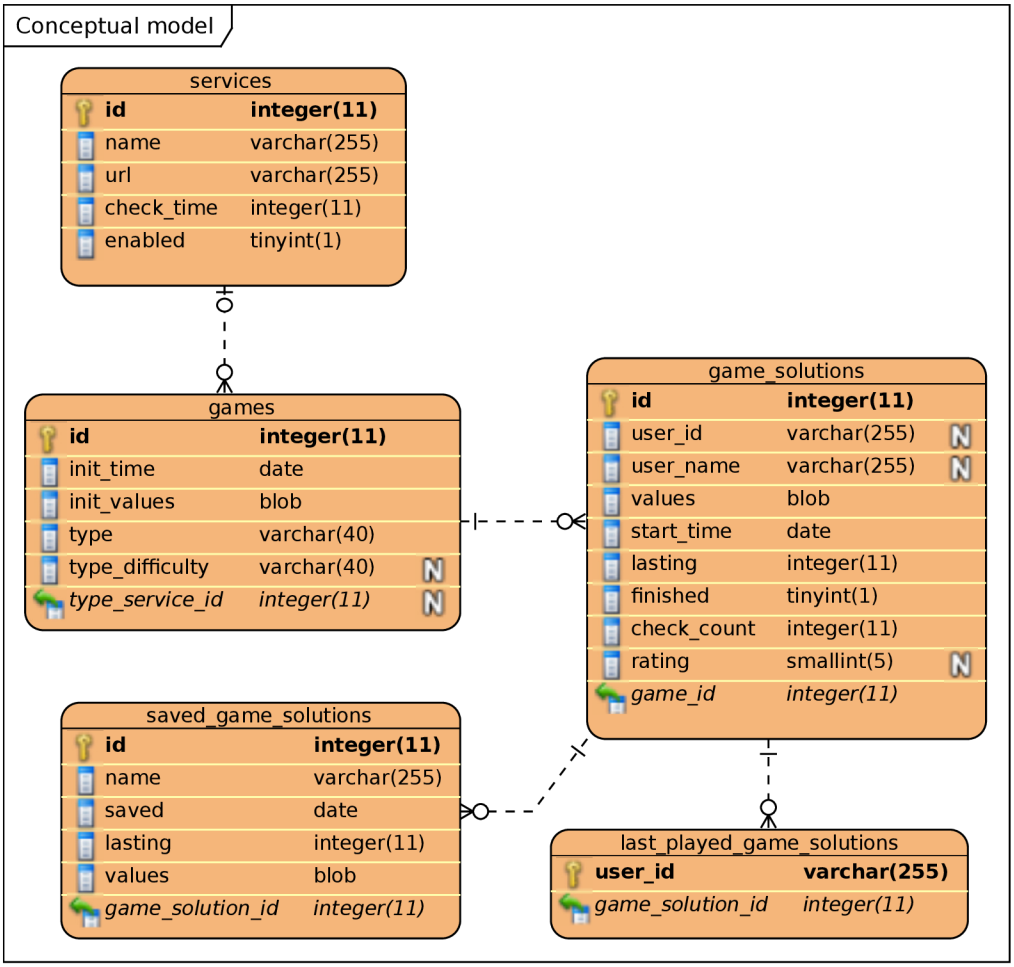


Figure 4.2: The conceptual model of data

Entities are managed by controllers that provide a set of allowed operations for this purpose. There is one controller for each entity. All controllers extend a `Controller` class that provides an access to persistent entities.

- The `GamesController` enables to create and to get Game entities, moreover it includes operations for getting of statistics.
- The `GameSolutionsController` creates, edits and gets Game solution entities. Entities may be filtered by their owner.
- The `SavedGameSolutionsController` creates and gets Saved game solution entities of users.
- The `LastPlayedGameSolutionsController` modifies and gets Last played game solution entities of users.
- The `ServicesController` creates, edits and gets Service entities.

Incoming communication of the Server side is not directly processed in a controller. There is another layer for dispatching of requests, processing of input data and creating of responses. The layer encases the functionality of controllers, moreover it adds other key capabilities, such as generating of games, obtaining of games from periodical remote services and checking of game solutions. There are several classes which create the layer, built according to the Facade design pattern.

- The `GameRestFacade` provides the generation of games and the encased functionality of the `GamesController`. The generation of game values is performed by the `Generator` class, as shown in the form of a UML sequence diagram in the Figure 4.3.
- The `GameSolutionRestFacade` provides the game checking capability and the encased functionality of the `GameSolutionsController`. Checking is performed by the method `check` of the `GameUtil` class.
- The `SavedGameSolutionRestFacade` encases the functionality of the `SavedGameSolutionsController`.
- The `LastPlayedGameSolutionRestFacade` encases the functionality of the `LastPlayedGameSolutionsController`.
- The `ServiceRestFacade` provides obtaining of games from periodical remote services and the encased functionality of the `ServicesController`.

The following text will be dedicated to obtaining of games from periodical remote services. Obtaining is realized by a driver that is acquired for a service from a factory. This architecture corresponds to the Factory design pattern. The factory class, called `PeriodicalServiceFactory`, contains the method `newDriverFor` with a Service entity which is a single parameter that the driver should be acquired for. The driver is chosen from a group of drivers whose names and class names are provided in the constructor of the factory class. Each driver must implement the `PeriodicalServiceDriver` interface that defines `init`, `isExpired` and `getGame` methods. A usage of these methods is shown in a form of a UML sequence diagram in the Figure 4.4. The `init` method attempts to initialize the driver to work with a service that is given by a parameter. If the method fails, the `IllegalStateException` will be thrown. The `isExpired` method checks whether the last obtained game of this service is expired. If the last obtained game is expired, a new game could be obtained by the `getGame` method. The `AbstractPeriodicalServiceDriver`

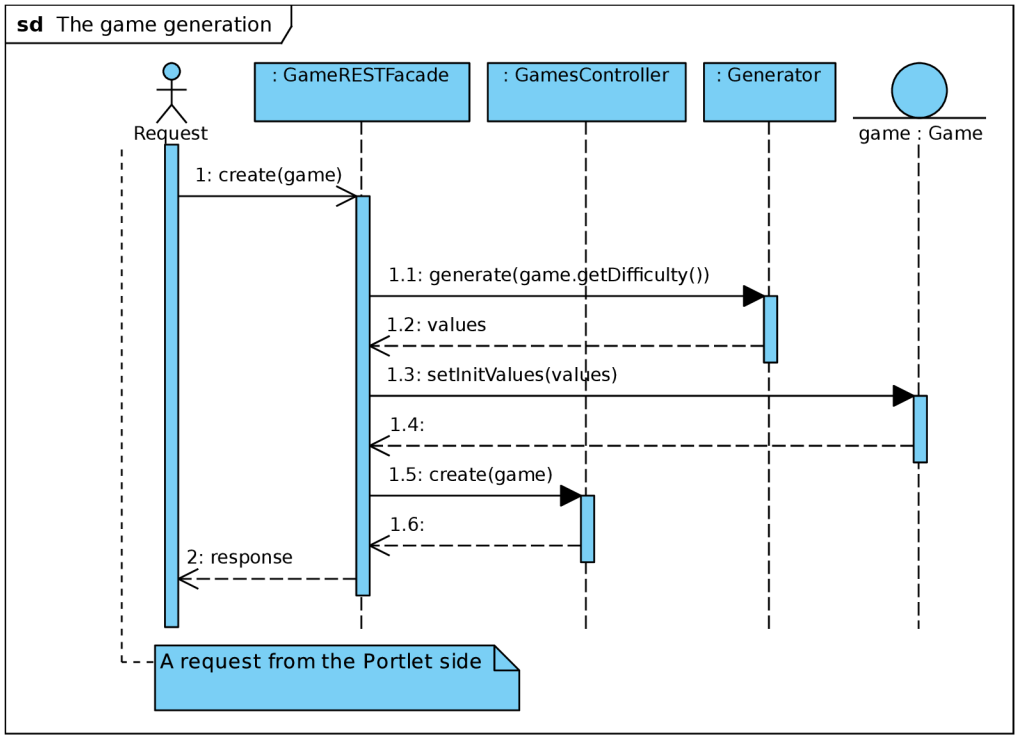


Figure 4.3: The sequence diagram of the game generation

provides an abstract implementation of the `PeriodicalServiceDriver`. A driver should extend this class and implement `getGame` and `isCapable` methods. The `isCapable` method checks if a service, given by a parameter, may be managed by the driver. This method is used in the `init` method. The `getGame` method takes a Game entity as a parameter that is a last obtained game of the service. The body of the method should obtain a game from the periodical remote service, it should then check if the game is equal to the last obtained game. If games are equal, the null value should be returned. In the opposite case an obtained Game entity should be returned. An implemented driver must be added to the configuration of the portlet application in order to be used. Configuration details are described in the section 6.2.1. The Figure 4.5 shows a UML class diagram of all these classes, moreover it contains an implemented driver, called `RahanWzCzPeriodicalServiceDriver`.

4.2.2 Client side

The Client side provides a logic for playing of games, a storage for the current state of a played game and a persistent storage for preferences of users. This part of application is made of a portlet which provides all these functionalities. The logic and the state is managed by a set of ECMAScript classes which are loaded by the portlet. The relation between classes may be seen in a form of a UML class diagram in the Figure 4.6. Classes manipulate with a fragment of a document, generated by the portlet, according to the current state of a game or user requests. The game state is stored periodically or after a change by creating a request to the Server side. Other operations which require stored data send requests to the Server side as well.

- The `Game` class is a basic element of the game logic which directs the initialization,

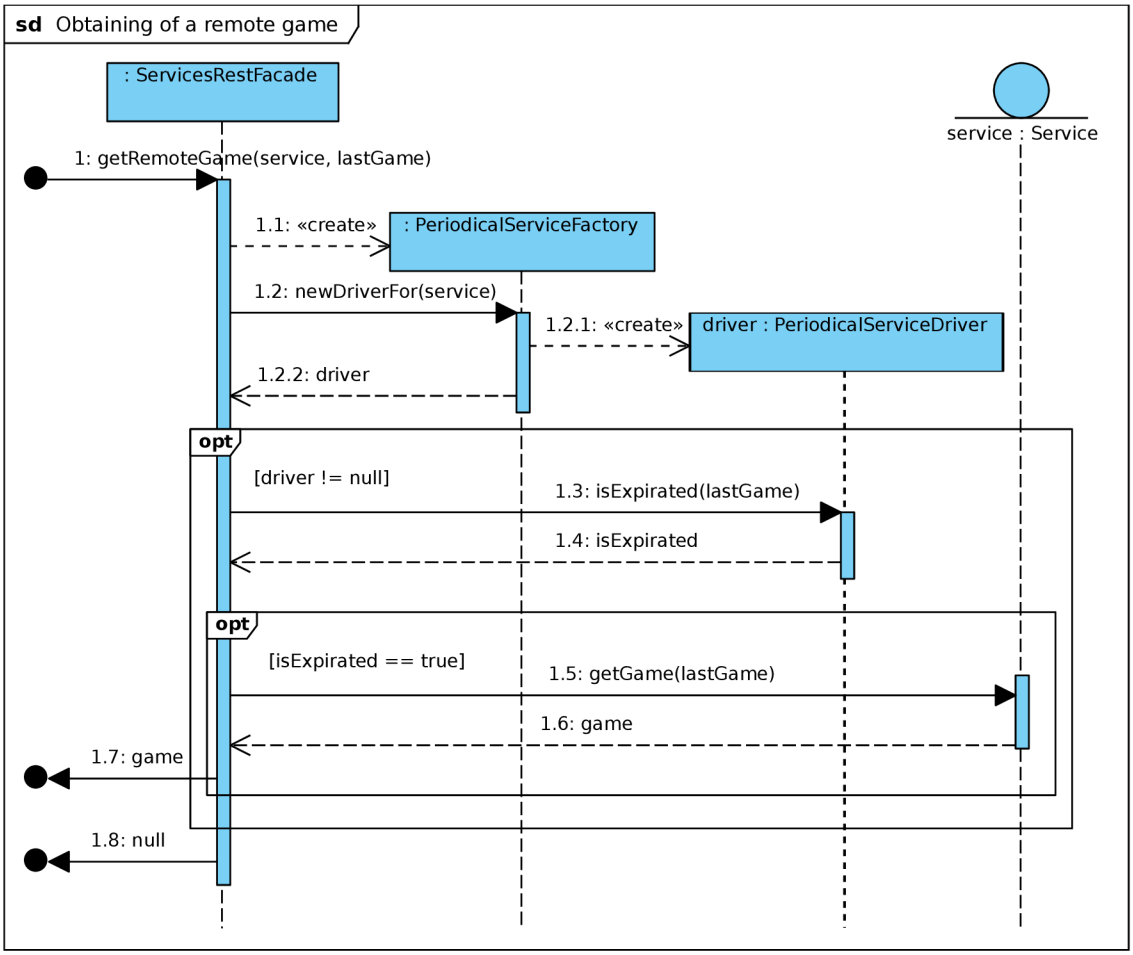


Figure 4.4: The sequence diagram of the `getRemoteGame` procedure for obtaining of remote published games

starting, resetting, storing and ending of a game.

- The `GameTimer` class is a timer for measuring of lasting of a game. The timer may be stopped and then started again.
- The `GameBoard` class contains a set of game board fields and provides an easy access for setting and accessing of their values. The game board may be disabled if playing of the game is forbidden.
- The `GameBoardField` class represents a field of a game board with a blank or a filled value which may be fixed.
- The `GameToolbar` class provides functionalities of saving, loading, checking, resetting, etc. The class includes a group of buttons which are available according to the game state. Each button has a predefined function which mediated one of listed functionalities. Typically, when a user presses a button, a request to the Server side is made. Received data are processed in the function of the button as well.

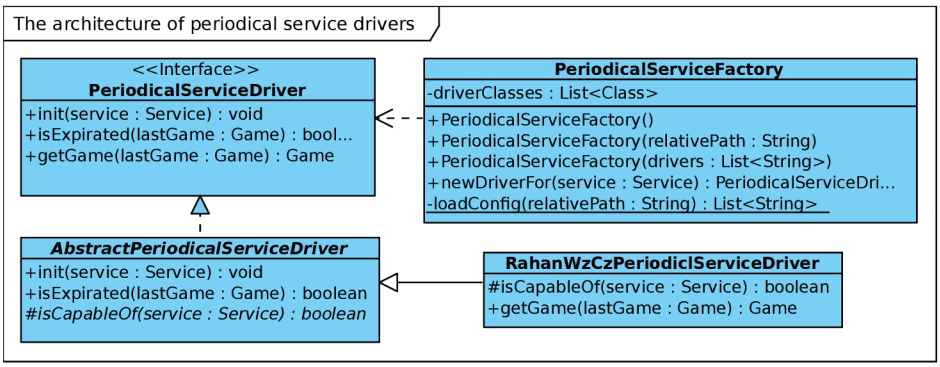


Figure 4.5: The class diagram of the architecture of drivers for obtaining of games which are published on periodical remote services

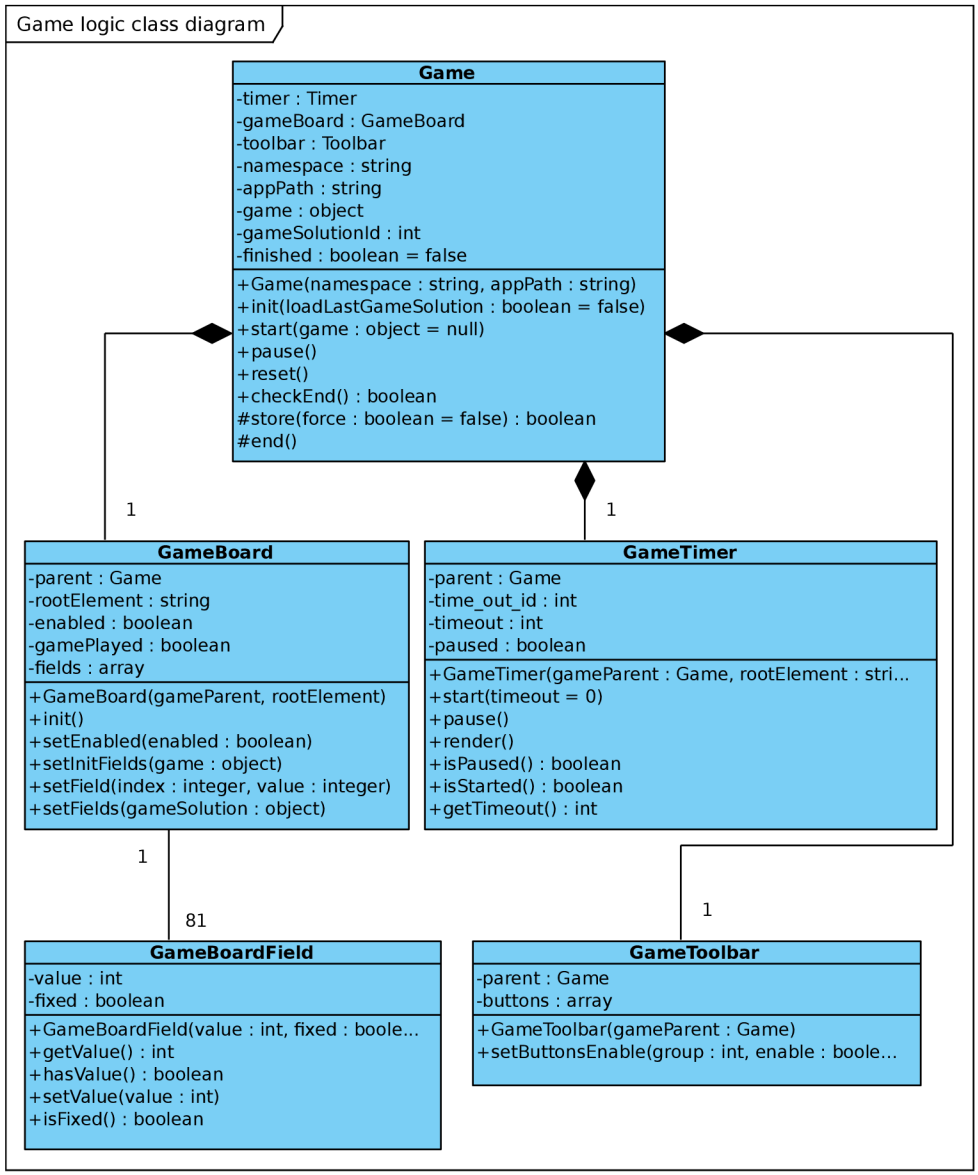


Figure 4.6: The class diagram of classes, related to the game logic

Chapter 5

Implementation

The implementation of the portlet application was made according to JSR 286, with a usage of other technologies. The application is a single project that contains the implementation of the Server side and the Client side.

The Client side is implemented as a portlet that represents the user interface (UI) of the portlet application. The portlet is defined in the `SudokuPortlet` class which extends the `GenericPortlet` class. The portlet supports view, edit and help modes by overriding of `doView`, `doEdit` and `doHelp` methods. Users are distinguished, using data, obtained by the `getRemoteUser` method. The user customization, such as a skin of the game board is stored in portlet preferences. The portlet does not directly communicate with the Server side. The communication is made by JavaScript scripts which are included in the generated fragment of the portlet content. More details about the portlet and its UI may be found in the following section.

The Server side is implemented separately from the portlet, as described in the proposal of the portlet application. Techniques which were used during the implementation of the Server site are discussed in the section 5.3.

5.1 User interface

The UI of the portlet is created by fragments of HTML documents that are styled by CSS[1] styles. The interaction between a user and the UI is done by a group of JavaScript scripts. Data are obtained from the Server side, using AJAX, and they are processed by JavaScript scripts as well as the interaction is. The application uses the Humanity[2] icon theme.

5.1.1 View mode

An application for playing of a Sudoku game is a part of the view mode of the portlet. The view mode contains the game board, the game toolbar and statistics. The interface in this mode adapts itself to the window state of the portlet. If the window state is normal, the game board with the game toolbar or statistics will be viewed. A user may switch manually between both options. If the window state is maximized, all components will be displayed.

The game board is automatically resizable according to a width of the portlet. Users may control the game board, using the mouse or the keyboard. Both control styles may be

[1] Cascading Style Sheets
[2] https://launchpad.net/humanity

combined. Controlling through the keyboard is based on moving around the game board with arrows. A value is directly typed into a focused game board field. Controlling through the mouse is made by using a hint that is shown after a mouse click on a game board field. The hint is a menu with possible values of the field. After clicking on a menu item a value of the item is set as a value of the field and the hint is closed.

The game toolbar mediates actions, such as creating of new games, saving of the current game, etc. A group of dialogs is created for purposes of these actions. For example, a dialog for creating of a new game is a wizard which allows generating of a new game or loading of a previously played game or loading of games from periodical remote services.

5.1.2 Edit mode

The edit mode enables changing of a skin of the game board and managing of periodical remote services.

Changing of a skin is enabled to logged user who may choose from a set of predefined themes or create an own custom theme. The current theme of a user is stored in portlet preferences. Predefined themes are specified in the configuration of the portlet application. This configuration is specified in the section 6.2.1.

Managing of periodical remote services is only restricted for portal administrators who may view all services in a separated tab and add or edit them by a dialog.

5.1.3 Help mode

The help mode contains a link to a user documentation. The user documentation is a PDF document that explains the UI of the portlet application.

5.2 Algorithm for generation of Sudoku games

The generation of Sudoku games is carried out by the Backtracking algorithm. A solved game is generated by the algorithm and then several solved values are blanked according to a level of a game difficulty. The generation is always performed on the Server side.

The Backtracking algorithm is a method of deep searching, related to the brute-force search technique but it only enumerates possible candidates of a solution. If the current candidate does not comply, it will go backward a try a different candidate until it finds a proper solution [2].

5.3 Used technologies

All technologies which were used during the development process of the portlet application are noted in this chapter. The whole project was managed by Apache Maven tool. Technologies such as JavaScript, AJAX, jQuery and Selenium IDE were used in the Client side and Java Persistence API, REST, JUnit, JSoup were used in the Server side. Details about technologies and their usage in the portlet application are listed in following sections.

5.3.1 Apache Maven

Apache Maven is a software tool for managing, building and deploying of software projects. A managed project is specified with its dependencies, build order, required plug-ins and

directories in a form of a Project Object Model (POM) which is an XML document, located in a root directory of the project in the `pom.xml` file. There are other tools for described purposes but Maven was chosen because it is capable of dynamic loading of libraries from remote or local repositories. This quality eases the development process of developed applications [6].

5.3.2 Java Persistence API

The Java Persistence API (JPA) is a part of the Java EE platform for managing of persisted data from relational databases. Schemes of databases are mapped to objects, called entities. An instance of an entity represents a row of a database table. Columns of a database table are mapped to properties of the mapped entity object, using annotations [24]. The Listing 5.1 shows the Service entity class with an identificator column and a name column which is mapped to a table, called services.

A group of entities may be managed by a single instance of the Entity Manager, defined by a persistence unit. A persistence unit contains connection information to a database, class names of entities, related to this connection, and configuration options of the Entity Manager. Persistence units are stored in the `persistence.xml` file. An initialized Entity Manager provides API for creating, updating, removing, finding and querying over entities, specified in the persistence unit. A database does not have to be created before using of entities or the Entity Manager. The Entity Manager can create a database according to entities. The creation of a database is configurated by an option of the persistence unit [15].

```
1 @XmlRootElement @Entity @Table(name = "services")
2 public Service implements Serializable {
3     @Id @GeneratedValue(strategy = GenerationType.IDENTITY) @Column(name = "id")
4     private Integer id;
5     @Basic(optional = false) @Column(name = "name", nullable = false)
6     private String name;
7     // ... other properties, constructors, getters and setters
8 }
```

Listing 5.1: An example of an entity class

For purposes of querying over entities the Java Persistence Query Language (JPQL) was created. The JPQL is similar to the SQL[3] language but it does not work directly with database tables and columns. A JPQL query operates over entities which may be also returned as responses. An advantage of the JPQL over the SQL is that the same JPQL query may be run on different database engines [24].

The JPA was used in controller and entity classes of the Server side which are described in the section 4.2.1. A part of the Service entity class is shown in the Listing 5.1. Another example, placed in the Listing 5.2, shows how the entity may be added to the database by the Entity Manager. Typically, the source code of the Listing is similar to the body of a method of a controller.

The portlet application defines two persistence units. The first unit is used for testing of controllers and entities, as described in the section 6.1.1. This testing unit contains a connection to a HSQLDB database, stored in the memory. The second unit provides a connection to a HSQLDB database as well as the previous unit but this database is a

[3] Structured Query Language

default storage of data of the application. Furthermore it is persistently stored in a file which is automatically created during the first running of the portlet application.

```
1  EntityManagerFactory emf = Persistence.createEntityManagerFactory("unitName");
2  EntityManager em = emf.createEntityManager();
3  try {
4      em.getTransaction().begin();
5      em.persist(new Service(null, "Rahan.wz.cz"));
6      em.getTransaction().commit();
7  } catch (Exception ex) {
8      em.getTransaction().rollback();
9  } finally {
10     em.close();
11     emf.close();
12 }
```

Listing 5.2: Persisting of a Service entity by the Entity Manager

A usage of the JPA eases following parts of a development process of the portlet application: the deployment, testing and a possible migration to other deployment environments. The HSQLDB database engine is not required. It can be easily changed to another suitable database engine in the configuration of persistence units.

5.3.3 REST

Representational state transfer (REST) is a data-oriented software architecture which consist of clients and servers. A client makes a request for data, located in a URI,[4] over the HTTP protocol and a server responds with data or a state message. The communication is stateless and cacheable. Requests are distinguished according to a URI and a HTTP method. Each HTTP method is used for another action: GET for getting of data, POST for creating of data, PUT for modifying of data and DELETE for removing of data [26].

RESTful applications may be built by a mechanism, specified in the JAX-RS. The JAX-RS is a specification of API for creating of RESTful Web Services in the Java. This concept is based on mapping between methods of Java classes and requests. Mapping is described by a set of annotations. A mapped Java class contains an annotation `@Path` that specifies a relative path of requests, processed by this class. A path is relative to a URI that is specified in the web deployment descriptor. Each method may define its relative path to a path of class with possible parameters, a HTTP method that may be processed and sets of consumed and produced types of data. The JAX-RS allows to transfer serialized Java objects with usage of the JAXB.[5] Objects are automatically transformed according to the definition of consumed or produced types of data [8].

An example of a class with two methods is shown in the Listing 5.3. The first method, called `create`, is mapped to a path `./game/` and to POST HTTP method and it consumes a JSON[6] document which is transformed to an object of the `Game` class. After processing of the object a response with a state is produced. On the other hand the `find` method is mapped to GET HTTP method and to a path with a parameter and it does not consume any data. For example, this method may be invoked by a request to the `./game/1` path.

[4] Uniform Resource Identifier
[5] Java Architecture for XML Binding
[6] JavaScript Object Notation

The method produces an object of the `Game` class, transformed to an XML or a JSON document according to the HTTP Accept Header of the request.

```
1 @Path("game")
2 public class GameRestFacade {
3     @POST @Consumes({"application/json"})
4     public Response create(Game game) { /* ... */ }
5     @GET @Path("{id}") @Produces({"application/xml", "application/json"})
6     public Game find(@PathParam("id") Integer id) { /* ... */ }
7 }
```

Listing 5.3: An example of a usage of the JAX-RS.

The portlet application is designed to be compatible with REST. Classes with name which ends with `RestFacade`, specified in the section 4.2.1, are implemented according to the JAX-RS, as described in the previous paragraph. These classes create a REST server which provides an access to data and functionalities of the Server side. A role of a REST client is represented by the Client side.

5.3.4 JavaScript

JavaScript is a scripting language from a family of ECMAscripts, designed to add an interaction to web pages. Commonly, a script which is written in JavaScript is embedded directly in a HTML document and it manipulates with the content and the structure of the document [25].

JavaScript contains a format for a representation of data, called JavaScript Object Notation (JSON). JSON provides an easy way for interchanging of data in text-based protocols, such as the HTTP [1].

The user interface of the Client side of the portlet application is partially written in JavaScript with a usage of AJAX and the jQuery library. Data between the Client side and the Server side are transmitted in the JSON format.

5.3.5 AJAX

Asynchronous JavaScript and XML (AJAX) is a group of web-based technologies which allow receiving, processing and displaying of data from a server by a client without refreshing of the whole web page. Typically, data are received in a form of an XML or a JSON document, then a JavaScript program modifies the DOM[7] of the web page according to received data [18].

AJAX was used for a communication between the Client side and the Server side in order to provide a fluent user interface of the Sudoku game.

5.3.6 jQuery

jQuery is a JavaScript library which simplifies JavaScript programming, document traversing, event handling, AJAX interactions, etc. Except for these qualities, jQuery provides a plug-in architecture that allows to add a huge amount of additional functionalities for web applications [16].

jQuery is used with several plug-ins in the Client side of the portlet application alongside with JavaScript, as described in the section 5.3.4. Used plug-ins are:

[7] Document Object Model

- jQuery UI[8] plug-in provides various kinds of widgets, such as dialogs, tabs, etc.
- Color Picker[9] plug-in represents a widget for selecting of a colour.
- DataTables[10] plug-in creates data tables with many features.
- jWizard[11] plug-in is used for wizard-like dialogs.
- Raty[12] plug-in provides a star-rating functionality.

5.3.7 JSoup

JSoup is a Java library for downloading, extracting and manipulating with a DOM of HTML documents. The main feature of the library is a capability of browsing with selectors which are similar to selectors, used in jQuery and CSS [9]. The portlet application uses JSoup API functions in order to obtain data from periodical remote services. The code in the Listing 5.4 shows how easily may data be extracted from a HTML document with the JSoup library.

```
1 Document doc = Jsoup.connect("http://rahan.wz.cz/daily_sudoku.php").get();
2 Elements inputs = doc.select("table tr td input");
3 for (Element input : inputs)
4     System.out.println(input.attributes().get("value"));
```

Listing 5.4: An example of a usage of the JSoup library.

5.3.8 JUnit

JUnit is a framework for automated unit testing. Typically, a unit test checks a method of a class and all unit tests of all methods of a class create a test case. This test case is encased to a Java class which may be run repeatedly. Commonly, test cases are automatically run after successful building of the application for verification of its functionalities [3]. The usage of JUnit in the portlet application is described in the section 6.1.1.

5.3.9 Selenium IDE

Selenium automates web-browsers by test cases that enable testing from the point of view of the user interface of the web application. Selenium IDE is a plug-in for Mozilla Firefox[13] that may be used for creating and running of test cases. It records activities of a user on a web page. A recorded activity is stored to a file and it runs repeatedly as a test case [20]. The description of a usage is placed in the section 6.1.2.

[8] http://jqueryui.com/
[9] http://www.eyecon.ro/colorpicker/
[10] http://datatables.net/
[11] https://github.com/dominicbarnes/jWizard
[12] http://www.wbotelhos.com/raty/
[13] http://www.mozilla.org/en-US/firefox/

5.4 Possible improvements

The current implementation may be improved in issues, such as the security and the internationalisation.

The security issue is related to the communication between the Client side and the Server side. The communication is not protected and it does not use any authentication. The protection of the communication may be made by replacing the HTTP protocol by the secured HTTPS protocol but this replacement depends on capabilities of the deployment environment. The authentication of users and their requests is more important issue because in the current implementation a user may change data of other users by the manual creation of requests to the Server side. There are more possible solutions. The first solution may be made by adding of some authentication mechanism to the current situation where sides of the communication are separated. The second solution rests on using of the serve resource mechanism from the portlet specification. This solution solves the problem but brings another one. The whole code from all REST facade classes would be aggregated to the portlet class without any possibility to use the mechanism, specified by the JAX-RS specification that would be unproductive. The best as well as the hardest solution is to extend the portlet specification in order to provide the functionality of the JAX-RS specification in the resource serving mechanism. The `serveResource` method would be complemented by a set of annotations for mapping of requests and by the possibility of dispatching of requests to other methods outside the portlet class.

English is the only language of the user interface at this moment. The internationalisation of the portlet may be implemented by the mechanism, specified by the portlet specification.

Chapter 6

Testing and deployment

The portlet application was continuously tested in order to ensure its functionalities and to simplify its deployment to environments, defined in the specification of requests. The portlet application was designed to be configurable without any intervention to source codes. The configuration helps during the migration and the customization of the portlet application to deployment environments.

6.1 Testing

Testing as well as the architecture of the portlet application was separated to individual parts. The Server side was tested, using JUnit tests and partially Selenium IDE tests that were used for testing of the Client side.

6.1.1 JUnit tests

The first group of unit test suites was designed to verify the generation of games and other small utilities of the portlet application, for example checking of the correctness of game solutions.

The second more extensive group is designed to test all controllers and their operations over entities by several test cases that test all of them together. For these purposes a separated data source in a form of a HSQL database, stored temporarily in the memory, is automatically created during testing.

Tests are located at the `/src/test/java` path and they may be run simultaneously by Apache Maven, using the CLI[1] command `mvn test` that is triggered in the root directory of the portlet application.

6.1.2 Selenium IDE tests

Several Selenium IDE tests were created to simulate basic user activities in the portlet application. Tests are useful during the development of the user interface, especially for functionality, implemented in JavaScript. Due to the fact that the user interface is quite extensive, manual testing would take a long time. On the other hand Selenium IDE tests are finished in a matter of seconds. The REST part of the portlet application is verified during tests as well as the user interface.

[1] Command-line interface

The Selenium IDE has its issues, given by the nature of this tool. Tests are bound to a web page which means if a change in the DOM of the web page is made, the test may be broken. Due to this issue, Selenium IDE tests were not used as JUnit tests but only for testing during the implementation process.

6.2 Deployment

The portlet application was ported to all environments, specified in the analysis. The deployment process, such as building, testing, packaging and deploying are made by using of Apache Maven.

The deployment process is customized by various configuration options. Furthermore it depends on the application server that is bound to the used portal. Currently, the portlet application is provided for Apache Tomcat and JBoss AS application servers. Building, testing, packaging and deploying is made by a single command. The command for JBoss AS is shown in the Listing 6.1.

```
1 mvn clean package jboss:hard-deploy -P "jboss-as" \
2           -Denv.JBOSS_HOME=<path to JBoss AS home directory>
```

Listing 6.1: The command for launching of the portlet application on JBoss AS.

After triggering of the command the portlet may be added to a portal page. In GateIn portal the portlet must be imported in the Application Registry section and then it may be added to a portal page.

6.2.1 Configuration

A configuration of the Client side rests on following items.

- Periodical remote services may be disabled in the `remotePublishersEnabled` initialization parameter of the portlet deployment descriptor which is located in the `/src/main/webapp/WEB-INF/portlet.xml` file.
- Drivers of periodical remote services are defined in the XML document, located in the `/src/main/webapp/WEB-INF/sudoku-portlet-periodical-service-drivers.xml` file. Drivers that are specified in this document are available to be used for services which are managed in the edit mode of the portlet by an administrator.
- Predefined skins of the game board are located in the `/src/main/webapp/WEB-INF-/sudoku-portlet-skins.xml` file. One of these predefined skins may be set as the default skin in order to provide a skin that is optimized to the appearance of the portal.

The Server side is bound to the application server of the portal at a path, defined in the `/src/main/webapp/WEB-INF/jboss-web.xml` file for JBoss AS and in the `/src/main-/webapp/META-INF/context.xml` file for Apache Tomcat. The default path is `/sudoku-game`.

Without any intervention to the configuration, the portlet application uses a HSQL database, stored in a file. The data source may be changed to another database, using the `/src/main/setup/jboss-ds.xml` file on JBoss AS and the `/src/main/webapp/META-INF-/context.xml` file on Apache Tomcat. Moreover the dialect of a new database source must be set in the `/src/main/resources/META-INF/persistence.xml` file on both application servers.

Chapter 7

Conclusion

The question of portals, introduced in the first chapter, is quite extensive. Important principles and properties of portals are explained in the second chapter that also contains basic information about the categorization of portals and characteristics of basic elements of the portlet environment. Furthermore it includes the description of the Enterprise portal which is the most common portlet type. The second chapter continues with the section about an open-source portal solution, called GateIn portal. The last part of the chapter is dedicated to portlets that are components of portals. Each portal contains portal pages which are occupied by these components in order to create their content. The communication between portlets and the portal is mediated through the portlet container, described in this chapter as well. The development of portlets is defined by two specifications. The third chapter describes these specifications. The first specification, published as JSR 168, defines the life cycle, the configuration, states, modes, preferences and sessions of the portlet. JSR 286 is a complement for the first specification and it adds capabilities, such as the coordination between portlets, resource serving, portlet filters and the caching mechanism. Content of both specifications is explained in detail and the most important parts are shown in examples.

The aim of the practical part was to develop a portlet implementation of the Sudoku game for Red Hat company which may use it as an exemplary portlet in GateIn portal. The analysis of the portlet application is placed in the fourth chapter. The implementation should provide the generation of games on demand, obtaining of games from periodical remote services, statistics of playing, etc. These capabilities are summarized in the specification of requests section. Before a next phase of the development available solutions were explored and some of them were chosen as periodical remote services. After the analysis a proposal, described in the fifth chapter, was designed. The architecture that is designed in the proposal separates the portlet application to two parts, the Server side and the Client side. The Server side persists games and other required data. Moreover it answers to requests for these data from the Client side which is a portlet, developed according to portlet specifications. The portlet provides the view portlet mode for playing of games, the edit portlet mode for the customization and the management of periodical remote services and the help portlet mode for accessing to the user documentation. Implementation details are placed in the sixth chapter that also contains a list of used technologies with their basic characterisation and possible improvements of the current implementation. Deployment instructions and methods of testing of the portlet application are described in the last chapter of the thesis.

The final version of the developed portlet implementation of the Sudoku game was submitted to Red Hat company for the inclusion into GateIn portal. The application may

be improved, as suggested in the implementation chapter. Furthermore the thesis will be publicly available in order to provide a tutorial for developers who start to develop portlets.

Bibliography

[1] *Introducing JSON* [online]. Available from: http://www.json.org/ [Accessed 2012-04-13].

[2] B. Antal. *The Backtracking Algorithm Technique* [online]. Available from: http://www.devarticles.com/c/a/Development-Cycles/The-Backtracking-Algorithm-Technique/1/ [Accessed 2012-04-05].

[3] K. Beck. *JUnit pocket guide.* Pocket References Series. O'Reilly, 2004. ISBN 9780596007430.

[4] C. Bishop. *What is a Web Application Server?* [online]. Available from: http://www.resultantsys.com/index.php/general/what-is-a-web-application-server/ [Accessed 2012-02-14].

[5] H. Collins. *Enterprise knowledge portals: next-generation portal solutions for dynamic information access, better decision making, and maximum results.* AMACOM, 2003. ISBN 9780814407080.

[6] Sonatype Company. *Maven: the definitive guide.* O'Reilly Series. O'Reilly, 2008. ISBN 9780596517335.

[7] D. Cowar and Y. Yoshid. *JSR 154: Servlet Specification Version 2.4* [online]. Available from: http://www.jcp.org/en/jsr/detail?id=154 [Accessed 2012-02-15].

[8] M. Hadley and P. Sandoz. *JSR 311: JAX-RS: The Java API for RESTful Web Services* [online]. Available from: http://jcp.org/en/jsr/detail?id=311 [Accessed 2012-04-12].

[9] J. Hedley. *JSoup: Java HTML Parser.* Available from: http://jsoup.org/ [Accessed 2012-04-07].

[10] S. Hepper. *JSR 286: Java Portlet Specification 2.0* [online]. Available from: http://www.jcp.org/en/jsr/detail?id=286 [Accessed 2012-01-04].

[11] S. Hepper and A. Abdelnur. *JSR 168: Java Portlet Specification 1.0* [online]. Available from: http://www.jcp.org/en/jsr/detail?id=168 [Accessed 2012-01-04].

[12] S. Hepper and O. Koth. *What's new in the Java Portlet Specification V2.0 (JSR 286)?* [online]. Available from: http://www.ibm.com/developerworks/websphere/library/techarticles/0803_hepper/0803_hepper.html [Accessed 2012-03-08].

[13] T. Heute. *GateIn Portal vs JBoss Enterprise Portal Platform* [online]. Available from: http://jboss-epp.professional-blog.com/?p=28 [Accessed 2012-02-26].

[14] T. Heute, S. Mumford, and L. Texier. *GateIn User Guide* [online]. Available from: http://docs.jboss.com/gatein/portal/3.1.0-FINAL/user-guide/en-US/pdf/GateIn%20User%20Guide%20en.pdf [Accessed 2012-02-26].

[15] Sun Microsystems Inc. *Interface EntityManager* [online]. Available from: http://docs.oracle.com/javaee/5/api/javax/persistence/EntityManager.html [Accessed 2012-04-11].

[16] jQuery Foundation. *jQuery is a new kind of JavaScript Library* [online]. Available from: http://jquery.com/ [Accessed 2012-04-13].

[17] K. Mukhar, C. Zelenak, J.L. Weaver, and J. Crume. *Beginning Java EE 5: from novice to professional*. Expert's voice in Java. Apress, 2006. ISBN 9781590594704.

[18] S.D. Olson. *Ajax on Java*. Java Series. O'Reilly, 2007. ISBN 9780596101879.

[19] J. Ottinger. *What is an App Server?* [online]. Available from:http://www.theserverside.com/news/1363671/What-is-an-App-Server [Accessed 2012-02-10].

[20] Selenium Project. *Selenium Documentation* [online]. Available from: http://seleniumhq.org/docs/02_selenium_ide.html [Accessed 2012-04-26].

[21] T. Rourke. *Application server* [online]. Available from: http://searchsqlserver.techtarget.com/definition/application-server [Accessed 2012-02-12].

[22] N.M.L. Sood. *Sudoku Gems*. Diamond Pocket Books (P) Ltd. ISBN 9788128812989.

[23] A. Tatnall. *Web portals: the new gateways to Internet information and services*. Idea Group, 2005. ISBN 9781591404385.

[24] Y. Vasiliev. *Beginning Database-Driven Application Development in Java EE: Using GlassFish*. From Novice to Professional. Apress, 2008. ISBN 9781430209638.

[25] W3Schools. *JavaScript Introduction* [online]. Available from: http://www.w3schools.com/js/js_intro.asp [Accessed 2012-04-13].

[26] Wikipedia. *Representational State Transfer* [online]. Available from: http://en.wikipedia.org/wiki/Representational_State_Transfer [Accessed 2012-04-09].

Appendix A

Content of attached CD

The attached CD contains source codes of the implemented portlet application and a user documentation.

- user-guide.pdf – a user guide
- README – a basic description of the application and installation instructions
- pom.xml – a configuration of Apache Maven
- src – a directory with source codes
- target/site/apidocs – a directory with the Javadoc API documentation in HTML format

Appendix B

Samples of Sudoku portlet

Screenshots show the implemented portlet, deployed on GateIn portal.

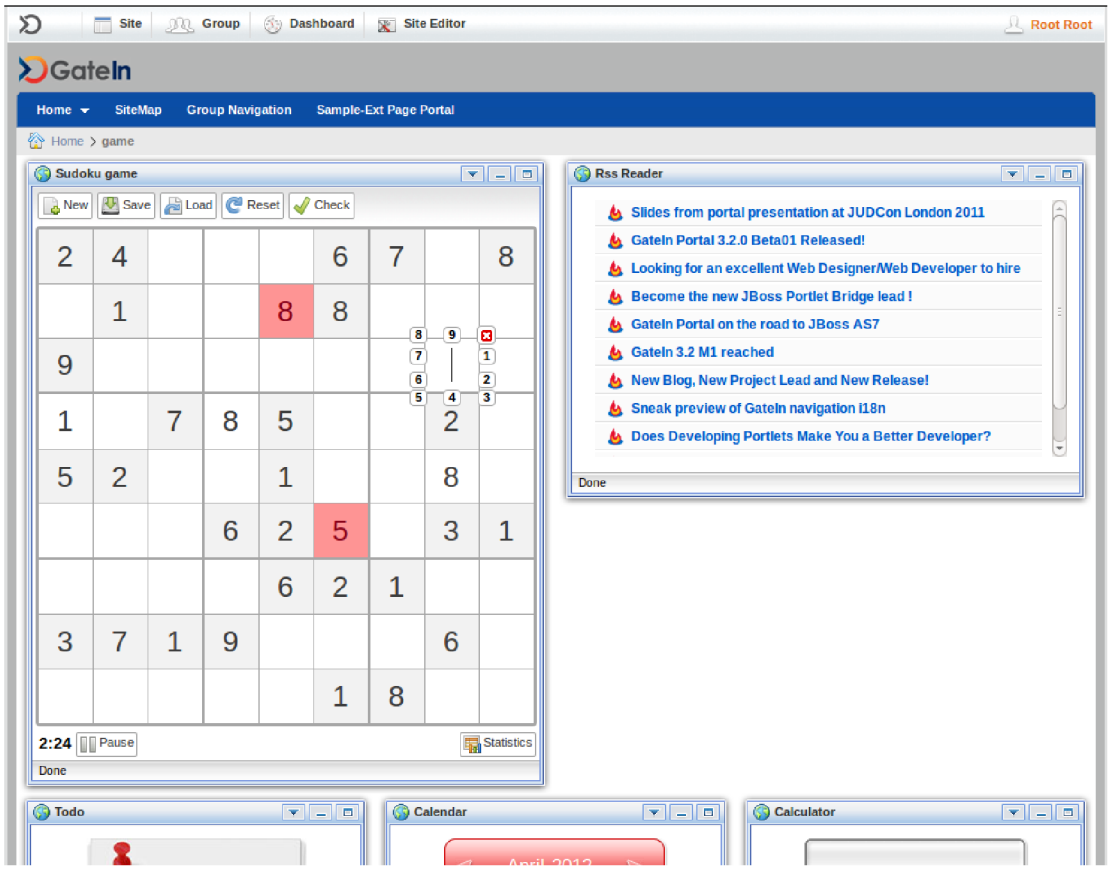


Figure B.1: The portlet in the view mode and in the normal window state.

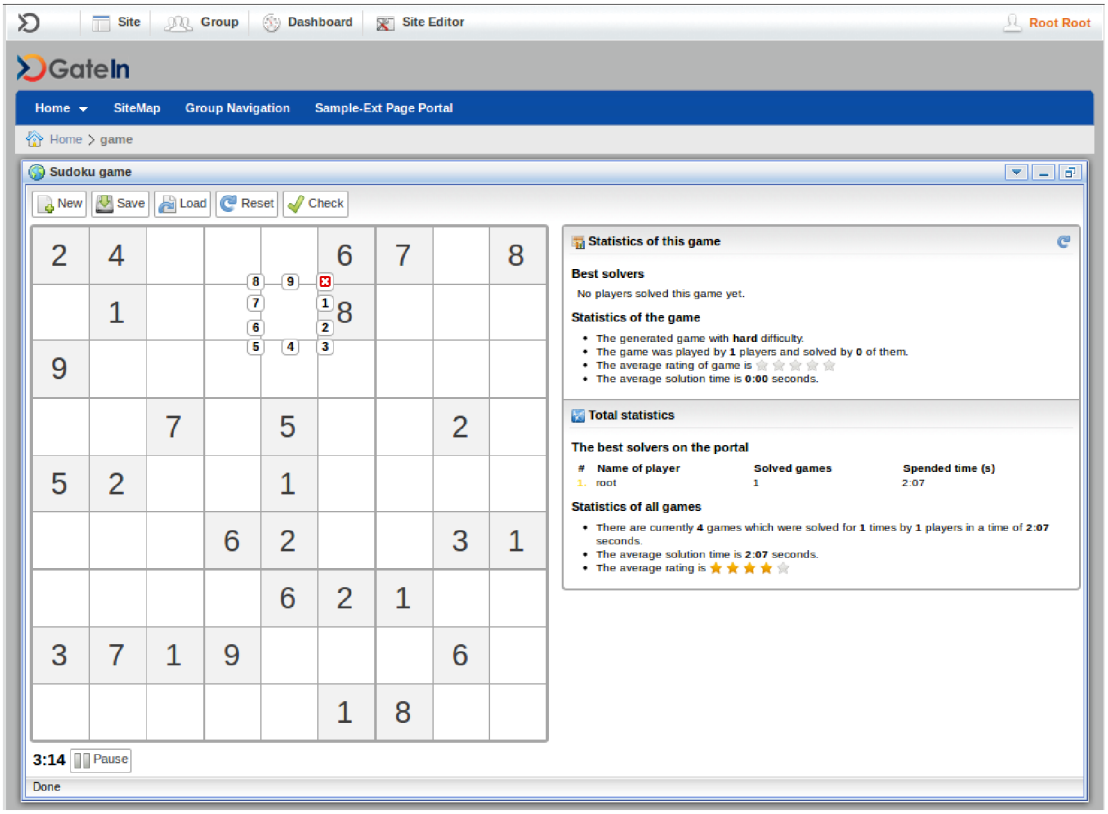


Figure B.2: The portlet in the view mode and in the maximized window state.

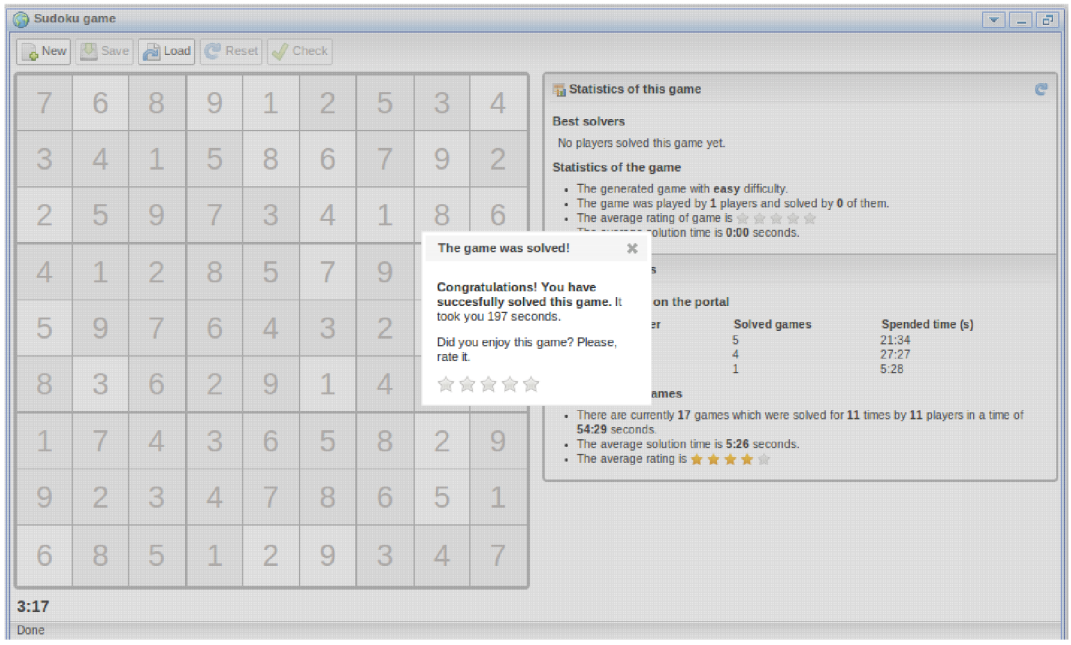


Figure B.3: The end of a game

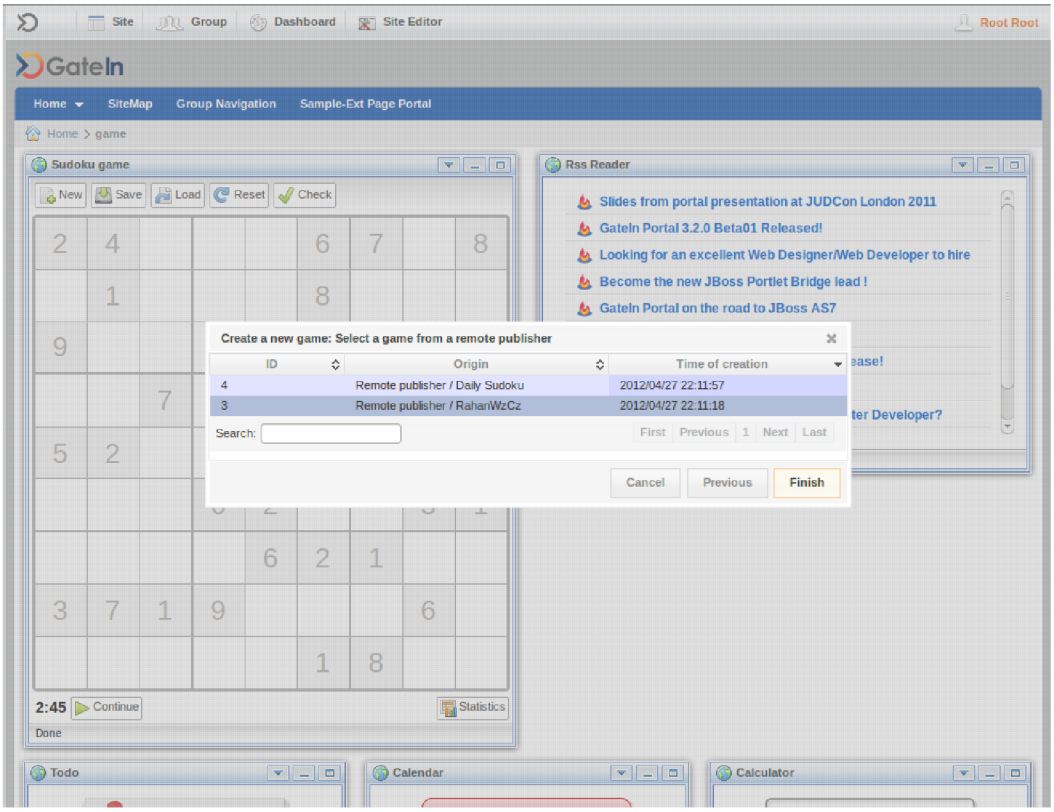


Figure B.4: Loading of a remote published game

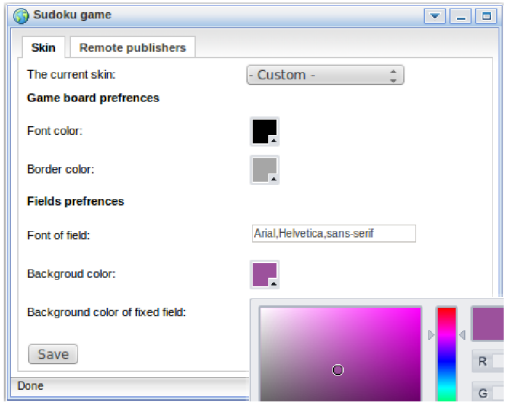


Figure B.5: The portlet in the edit mode during managing of a custom skin of the game board.

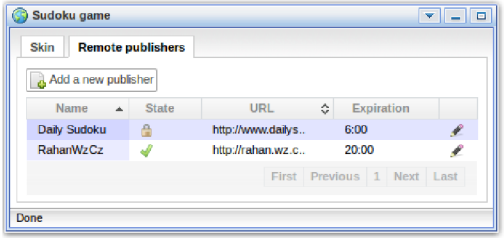


Figure B.6: The portlet in the edit mode during managing of periodical remote services.