



evropský  
sociální  
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,  
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání  
pro konkurenceschopnost



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

## Platforma průmyslové spolupráce

CZ.1.07/2.4.00/17.0041

### Název

Configuration Conversion JBoss AS5 to JBoss AS7

### Popis a využití

- popis evoluce aplikačních serverů
- výuka: pokročilá Java

### Jazyk textu

- anglický

### Autor (autoři)

- Roman Jakubčo

### Oficiální stránka projektu:

- <http://lasaris.fi.muni.cz/pps>

### Dostupnost výukových materiálů a nástrojů online:

- <http://lasaris.fi.muni.cz/pps/study-materials-and-tools>

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Description of JBoss Application Server</b>	<b>3</b>
2.1	Java Enterprise Edition Servers	3
2.2	Different Versions of the JBoss AS	3
2.3	The JBoss AS5 and Earlier Versions	4
2.3.1	JBoss AS 5 structure	4
2.4	Changes in JBoss AS 7	6
2.4.1	JBoss AS 7 Structure	6
2.4.2	Command Line Interface in JBoss AS 7	8
2.5	Configuration Files in Application Servers	9
<b>3</b>	<b>Analysis of Migration</b>	<b>11</b>
3.1	Configuration of Datasource Service	11
3.2	Configuration of Logging Service	13
3.3	Configuration of Security Service	14
3.4	Configuration of Resource Adapters	15
3.5	Configuration of Web Engine	16
<b>4</b>	<b>Application for Automated Migration</b>	<b>18</b>
4.1	Lifecycle of Application	18
4.2	Structure of Created Application	19
4.2.1	IMigrator Interface	20
4.2.2	Migrators Package	21
4.3	JAXB and EclipseLink MOXy JAXB Implementation	22
<b>5</b>	<b>Implementation of Migration</b>	<b>25</b>
5.1	Migration of Datasource Configuration	25
5.2	Migration of Logging Configuration	27
5.3	Migration of Security Configuration	28
5.4	Migration of Resource Adapters Configuration	28
5.5	Migration of Web Engine Configuration	29
<b>6</b>	<b>Conclusion and Use of Application</b>	<b>31</b>
6.1	Using the Created Application	31
6.2	Conclusion and Future	32

<b>Appendices</b> . . . . .	<b>35</b>
<b>A Attached files</b> . . . . .	<b>35</b>
<b>B Usage of Application</b> . . . . .	<b>36</b>
<b>C Configuration Examples</b> . . . . .	<b>37</b>
<b>D Examples of Created CLI Scripts</b> . . . . .	<b>48</b>

# 1 Introduction

JBoss Application server is a Java based application server developed by JBoss [15], which is now a division of Red Hat, Inc. The origin of JBoss is in 1999, when Marc Fleury started free software project EJB-OSS, which was primarily focused on implementing Enterprise Java Beans (EJB) specification from Java Enterprise Edition [16]. The name of the project had to be changed to JBoss because Sun Microsystems did not like use of their EJB trademark in the name of the project. Then in 2006 JBoss was acquired by Red Hat and became a division of the company, which specializes in development and support of the open source middleware software and one of these software products is JBoss AS. JBoss AS is licensed under GNU Lesser General Public License, which makes it an open source software [17]. Generally JBoss AS is very popular among business companies and has very strong community.

Almost all software products evolve and upgrade and JBoss AS is not different. Each version has different characteristics and, at the moment, the newest version of JBoss AS 7 is focused on usability and speed.

Different characteristics and new features lead to major changes in the structure and configuration of the application server. This creates a problem for companies with well-configured servers and their deployed applications. If they want to upgrade their servers, they must spend money and time on manual migration of the configuration to the newer version because the way of configuring can be completely different and it is required to understand and study the new version from the project's documentation. Therefore, companies rather use the older application servers, which are already running and still providing all the necessary functions than risking and spending money or losing time on upgrade. This situation creates a problem for the JBoss customer support and also depriving users of newer functions. At the moment, majority of companies are still using very popular JBoss AS 5, which has a completely different structure and ways of configuration than the newer version JBoss AS 7. Red Hat is trying to solve this problem and has several projects and guides for migration from previous versions. One of them is the migration guide to JBoss AS 6 for applications, which covers migration of the server configuration only slightly and still requires manual work. Some JBoss's projects like Iron Jamacar covers the automated migration of datasources, but not the whole configuration of the server and all its important services.

The problem of migrating configuration of JBoss AS 5 to JBoss AS 7 and finding solution for it is the first main goal of this thesis. The second goal is to create an application capable of automated migration. This application must be able to parse

configuration from JBoss AS 5, migrate AS5's configuration and apply it into fresh installation of JBoss AS7 using CLI, which was added as new feature in JBoss AS 7 for management of the application server. The third and important goal is that the application must be developed with the intention of a further development by community and an easy expansion of its functionality and features. The automated migration will be much more comfortable for users than changing several configuration files manually as it must be done now.

An application server is a quite complicated software with many services and it is a lot of work to migrate all services. Because of that, this thesis only covers migration of five important and the most common services customized by users in the application server with hopes for the future upgrades done by the community. Taking everything into the consideration, the application has a lot of potential for customers and the JBoss community.

This thesis contains six chapters. Chapter 2 introduces JBoss AS and its versions. It also describes structures of JBoss AS 5 and JBoss AS 7 along with their main differences. In chapter 3 we take a detailed look on ways of configuration of migrated services in both versions. Chapter 4 describes created application and JAXB MOXy technology, which is used for representation of configuration files as Java objects. Chapter 5 describes the process of migration of these services. Chapter 6 provides a summary of results and possible future of the project. Besides these chapters, there are also three appendixes. Appendix A describes possible commands and usage of the application. Appendix B provides examples of configuration files and their migrated representations created by the application. And the last Appendix C contains examples of CLI scripts corresponding to examples of services in Appendix B.

## 2 Description of JBoss Application Server

To understand what is the primary focus of this thesis and the developed application, first it is required to understand what is JBoss Application Server along with its history and different versions. Each version of an application server is usually configured differently than the others. For our purposes it is required to understand differences in the configuration of JBoss AS 5 and JBoss AS 7. Everything mentioned will be detailed in this chapter.

### 2.1 Java Enterprise Edition Servers

Before we describe a Java application server, we need to understand what exactly is an application server. An application server can be described as a container for applications or an environment where they should run. This container that provides required services to applications deployed to it. This means applications do not need to implement services like connecting to database, interacting with different clients, security and many more things, the applications may need.

The Java application server standardizes the application architecture. Developers can use defined component models to develop their components, which can be deployed to the application server by a standard deployment model. The server provides services to these running components. Since applications are working in a standardized framework, the services are available to applications transparently. It is only required to provide some sort of information to call services from applications, which is usually done by XML annotations. This system can reduce the integration code and configuration of deployed applications [3].

There are several other application servers besides JBoss AS. Other open source servers are JOnAS by Object Web, Geronimo by Apache, TomEE by Apache, Resin Java Application Server by Caucho Technology, and GlassFish by Oracle. Well known are also commercial servers WebLogic Application Server by Oracle and WebSphere Application Server by IBM [6].

### 2.2 Different Versions of the JBoss AS

All projects and programs evolve and are upgraded and JBoss AS is not different. Though, there are earlier versions of JBoss AS like JBoss AS 1.0 and JBoss AS 2.0, the first stable release was JBoss AS 3.0. Different versions of AS are based on different versions of Java EE specification. Until AS 7, the version number of the application

server referenced the edition of Java EE specification that the server implements. There are two implementations of Java EE 6, which are JBoss AS 6 and JBoss AS 7.

The focus of migration and this thesis is on JBoss AS 5 and JBoss AS 7. The first reason of this choice is that the AS7 is completely different than the AS5 in many ways, which will be more detailed further in this chapter. More importantly, the AS7 was released only few months after a major AS6 release, what is not very ordinary for major releases. This also means that the AS6 did not have time to spread like the AS5, which is still used by many users. Due to differences in mentioned versions, users have difficulties when moving their applications and configuration of their application server to the newest version. The solution to this problem can be an automatic migration.

### 2.3 The JBoss AS5 and Earlier Versions

Earlier versions of JBoss AS were based on Java Management Extension (JMX) microkernel. All services of the application server were written as Managed Beans (MBeans), which were plugged into the JMX kernel. This architecture enables easy adding and removing services in the application server.

However in JBoss AS 4.0.3 release there were the first signs of migration on a microcontainer architecture, which enables to write application server services using Plain Old Java Objects (POJOs). The AS5 has taken major steps on this path and the microcontainer is the visible part of the server architecture. Although the AS5 is based on a microcontainer kernel and it has a microcontainer architecture, the JMX still plays important part in the server [3, 1].

#### 2.3.1 JBoss AS 5 structure

To understand an application server, it is required to know the file structure of the application server. The structure of the AS5 is shown in Figure 1 and will be described in this section.

The `bin` directory contains scripts for starting and stopping the application server. The directory also contains other scripts for various purposes like accessing a JMX command line, discovering AS clusters and working with Web services. All scripts are available in Windows and Linux alternatives. The `client` directory contains client libraries required for running client's applications and they are needed for a communication with the application server.

These `client` libraries are loaded by the client application, which are called standalone clients or remote clients. Standalone clients use these libraries for remote calls of

the server-side part of the whole application.

bin	29 items folder
client	93 items folder
common	1 item folder
lib	99 items folder
docs	5 items folder
lib	53 items folder
server	5 items folder
all	9 items folder
default	8 items folder
conf	15 items folder
data	4 items folder
deploy	45 items folder
deployers	23 items folder
lib	0 items folder
log	3 items folder
tmp	14 items folder
work	1 item folder
minimal	4 items folder
standard	8 items folder
web	4 items folder

Figure 1: JBoss AS 5 Structure

The `docs` folder does not contain a documentation for as it may seem. In fact, it contains DTD files and XML schemas for configuration files of the server, configuration examples and licenses of libraries included in the application server.

The `lib` directory contains libraries required to start the server. The Microcontainer and JMX kernel are also stored here. This directory should not be changed. The `common` directory also contains a folder `lib`. The `lib` folder is a repository for shared libraries used by all application server configurations. Libraries, which should be used only by a specific server configuration, can be stored in its `lib` folder.

The `server` folder is the most important for the migration. As shown in Figure 1, it contains a configuration for different servers. Each one of them contains four folders `conf`, `deploy`, `deployers` and `lib`. These folders are responsible for configuration of all services of the application server and deployment of applications into the server. Configurations for different services are stored in their specific XML files in one of these folders. In addition, the server also creates few folders for temporary files and logs on the first start. These folders are `log`, `tmp`, `data` and `work`. [3, 1]



## 2.4 Changes in JBoss AS 7

The JBoss AS 7 has several major changes from older releases and new features. The first change is in the kernel of the application server, which is no longer based on Microcontainer. The AS7 has a completely new kernel, which is now based on two main projects:

- JBoss Modules – a standalone implementation of modular class loading [7] in Java and handles class loading in AS7
- Modular Services Container (MSC) – provides a way to install, uninstall and manage services used by the application server

These two projects create completely modular kernel and allow the server to have a modular architecture having completely changed the means of configuration of the application server – how and where can be server configured.

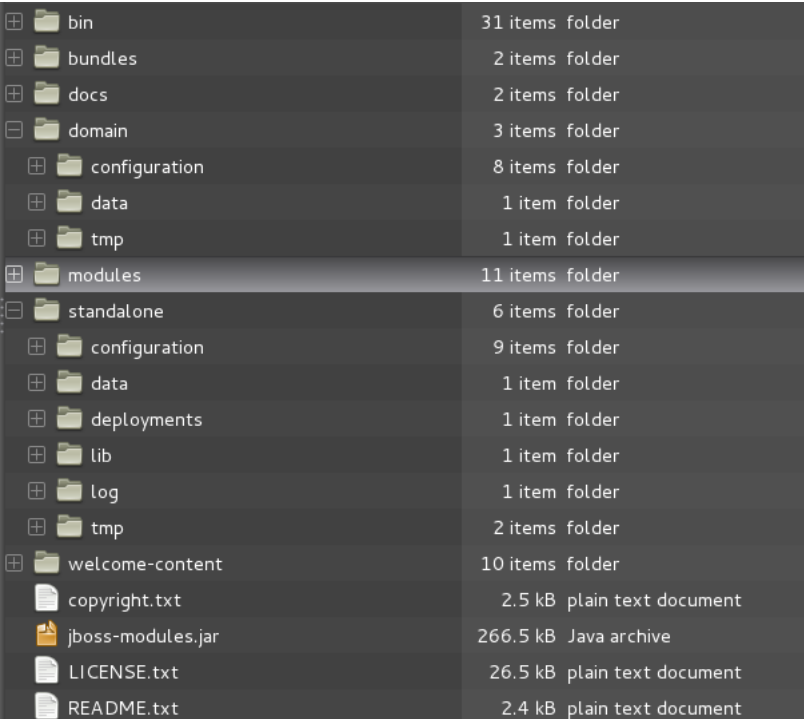
### 2.4.1 JBoss AS 7 Structure

The mentioned changes also affected the structure of the application server, but some folders have remained the same as in AS5. The `bin` and `doc` folders have the same functions as in AS5 structure with small differences. The `bin` folder contains new scripts and the `doc` folder contains only XML schemas and examples. Some of the new folders have identical function as other folders in the AS5, but, due to changes of the architecture of the application server, they have different names and structure. There are also three completely new folders with new functions for the AS7 and these folders are `bundles`, `domain` and `welcome-content`. The AS7 structure with all folders is shown in Figure 2.

The `welcome-content` folder contains the default web page for the server, which is served on root path by default. The `bundles` folder is used for storing OSGi (Open Services Gateway initiative) bundles.

The `modules` folder is a new modular structure for the `lib` directory in the AS5. All libraries of the application server are stored here, in a set of subfolders. The paths of folders are derived from names of modules. For example, `org.jboss.jboss-transaction-spi` module will be stored in `modules/org/jboss/jboss-transaction-spi` folder. Server's bootstrap library is stored in a root folder of the server in the `jboss-modules.jar`. This file is all what is required to bootstrap the new application server modular kernel.

The AS7 introduced a concept of server domains. Server domains are used for managing sets of instances of the application server organized in a domain. On the other



bin	31 items folder
bundles	2 items folder
docs	2 items folder
domain	3 items folder
configuration	8 items folder
data	1 item folder
tmp	1 item folder
modules	11 items folder
standalone	6 items folder
configuration	9 items folder
data	1 item folder
deployments	1 item folder
lib	1 item folder
log	1 item folder
tmp	2 items folder
welcome-content	10 items folder
copyright.txt	2.5 kB plain text document
jboss-modules.jar	266.5 kB Java archive
LICENSE.txt	26.5 kB plain text document
README.txt	2.4 kB plain text document

Figure 2: JBoss AS 7 Structure

hand, a standalone server is a single server, which is not part of a domain and it resembles a single server in older versions [2]. The configuration of a domain server is stored in the `domain` folder. The domain server is not part of the migration, so it is not required to describe it any further. The main focus is on the standalone server and its configuration.

The `standalone` folder is a configuration of the standalone server and it resembles server folder in the AS5. It has almost identical folders to the server subfolders with some changes. Configuration of services of the application server is now stored in the `configuration` folder. This folder contains few XML configuration files:

- `standalone.xml`
- `standalone-full.xml`
- `standalone-ha.xml`
- `standalone-full-ha.xml`
- `standalone-osi.xml`

Each file represents a different configuration of the server similar to server subfolders in the AS5. These files are the most important for the migration because all migrated services must be migrated to one of these files. [2]

#### 2.4.2 Command Line Interface in JBoss AS 7

The Command Line Interface (CLI) is the new feature of AS7 for administration and management of the application server. It is way how to manage the application server using command line. The strong advantage of using command line and scripts is a certain possibility to execute commands as part of batch or macros, which can be used for repetitive actions. The CLI shell is run by `jboss-cli` script located in `${JBOSS7_HOME}/bin` folder and it has many useful features. The CLI can be used for looking up settings for different services, changing them or adding new. It has several commands for different actions and it also provides auto completion for commands using the Tab key similar to Linux command shell. Different commands and useful tips can be found in AS7 documentation<sup>1</sup>.

The main function of the migration application is applying the migrated configuration of AS5 into standalone configuration in AS7. The problem is that it is only migration of configuration and not its validation. That means that even configuration, which has typo or other mistakes but still holds structure and syntax of the configuration file is migrated and imported into configuration file. The user can then find out that some of migrated services do not work only after start up of the application server. On the other hand, adding a configuration of services via CLI provides basic validation of processed data. That is one of the reasons for creating CLI scripts in the application.

There is also a possibility of using CLI management API in programs<sup>2</sup>. This way can be running server modified directly from the application. This API also provides batch functionality for commands, which means that if one commands fails the whole migration will rollback and the configuration of AS7 will stay unchanged.

Another reason is that the configuration file in AS7 can be changed in some minor way with the next version of the application server. This can destroy functionality of importing migrated configurations right into configuration file but the CLI scripts should remain compatible backwards in all new releases. This makes CLI a strong and important feature of the created application for use in the future. Examples of CLI scripts created by the application are shown in Appendix D.

---

<sup>1</sup><https://community.jboss.org/wiki/CommandLineInterface>

<sup>2</sup><https://docs.jboss.org/author/display/AS71/The+native+management+API>

## 2.5 Configuration Files in Application Servers

We mentioned in the section describing the structure of the AS5 that services configurations in the AS5 are stored in separate files that, in their turn, are stored in four main folders of the server. These files are all XML files with specific syntax and settings. At the moment, the application handles migration of these files and corresponding services:

- `-ds.xml` files responsible for configuration of datasources or resource adapters in the AS5. The server may have more than one of this file types.
- `jboss-log4j.xml` file responsible configuration of logging in the AS5. It is basically log4j configuration file commonly used in applications, which use the log4j API.
- `jboss-login.xml` file responsible for configuration of Java Authentication and Authorization Service (JAAS) or, in the other words, the security in the AS5.
- `server.xml` file responsible for the configuration of JBoss Web 2.1.x, which is based on Tomcat 6.x. This basically means that the `server.xml` is a configuration file for Tomcat server.

A detailed description and examples of all files will be shown in chapters describing migration of each service(subsystem) in a greater detail. It is also important to point out that the application at this moment does not support migration of cluster configuration of the AS5 and cluster configuration files. This feature may be added in the future.

The configuration of the AS7 is stored in one XML file, as mentioned previously in 2.4.1. The default file is `standalone.xml`. There are several others with different settings but all files have the same structure and syntax. A configuration file in the AS7 is not static. It means all changes made to the server when it is running are automatically reflected in it. A part of the automatic migration is importing migrated settings into these files. The structure of the configuration file is described in XML schemas for standalone file in `${JBOSS7_HOME}/docs/schema` or in documentation for the AS7 [9] or in book about AS7 [2].

The root element has seven main children elements. The most important for migration is `<profile>` element containing a collection of `<subsystem>` elements. The `<subsystem>` elements are responsible for the configuration of all services provided by the application server. There are several different subsystems defined in `<profile>` element. Each one of them is responsible for configuration of a different service of the application server and each has a `xmlns` attribute that defines the type and version of the subsystem.

For example, the logging subsystem is defined as `<subsystem xmlns="urn:jboss:domain:logging:1.1"/>`. A detailed description of each subsystem will be in chapters describing migration of few chosen subsystems.

Another element is the `<extensions>` element containing extensions for the application server. An extension is packed as module, which extends capabilities of the server and it is also stored in `module` directory. Extensions that should be available to the application server should be defined in this element.

Next element is `<system-properties>`, which is used for setting and adding system-wide properties for the application server and its configuration.

The `<paths>` element is intended for setting paths in the application server. The path defines logical names for file system paths, which then can be referenced by it. Several paths are already defined in the application server and cannot be overridden.

Another is `<management>` element containing definitions of security realms and management interfaces for the application server. These are used for adding users of the application server.

The `<interfaces>` element contains declarations of logical names for a network interface, an IP address or a host name to which sockets can be bound. These declared interfaces can be then referenced in other sections of the configuration file.

Another one is the `<socket-binding-group>` element, which contains declaration of *socket-bindings*. A `<socket-binding>` element defines the logical name for a socket (network port) that can be then referenced by it.

The last element is `<deployments>` element that contains a list of deployed applications on the application server. This element is always updated after deployment of application.

## 3 Analysis of Migration

Currently, the application migrates only five services of the application server that are changed by users the most often. Each service or subsystem has different functions and configuration options. These services and their configuration in both servers will be detailed in this chapter.

### 3.1 Configuration of Datasource Service

*Datasource* subsystem is one of the subsystems in the application server and it is responsible for configuration of *datasources* that are very important for most enterprise applications. Their function is to configure database connectivity and each *datasource* contains a pool of database connections. They are used by applications deployed on the server [2]. This makes the *datasource* subsystem the first choice for automated migration, which will be detailed in this chapter.

The configuration of *datasources* in AS5 is stored in specific XML files in directory `${JBOSS5_Home}/server/<server-name>/deploy`. These files are marked by `-ds.xml` suffix in the name of the files and the prefix is usually the name of the database of *datasources* declared in it. It is a common practice to store *datasources* for the same database in the same file. An application server can have several configuration files for *datasources*; AS5 supports several databases. Files with `-ds.xml` suffix can also contain a configuration for *connection-factories* having `<connection-factories>` root element that holds the configuration of resource adapters in AS5. For more information about connection factories and their migration refer to 3.4.

The `<datasources>` root element holds configurations of three different *datasources*:

- `<no-tx-datasource>` uses the `NoTxConnectionManager` service and this connection manager does not take part in JTA transactions [1]
- `<local-tx-datasource>` on the other hand uses the `LocalTxConnectionManager` supporting JTA transactions
- `<xa-datasource>` uses `XaTxConnectionManager`, which also supports JTA transaction but in addition to `LocalTxConnectionManager` supports two-phase commit

All three *datasources* have several identical children elements being responsible for their configuration and, of course, there are also specific elements for each type of *datasource*. Important elements are `<driver-class>` and `<xa-datasource-class>`, which reference classes in the JDBC driver. This driver is a JAR file that can be stored in

`${JBOSSE5_Home}/common/lib` or `${JBOSSE5_Home}/server/<server-name>/lib` directory. The driver's library is crucial for successful migration of *datasource*.

More information about configuration elements in all *datasources* can be found in a XML schema `jboss-ds.xml` stored in `${JBOSSE5_Home}/docs/schema`. There are also many examples for configuration files for different types of databases stored in `${JBOSSE5_Home}/docs/examples/jca`.

In AS7, configuration of a *datasource* is done in one subsystem of the configuration file. The *datasource* subsystem is provided by IronJacamar project<sup>3</sup>, which defines configuration possibilities of the *datasource* subsystem [12]. An example of the *datasource* subsystem is shown in Listing 9 in Appendix C.

The AS7 configuration has only two types of *datasources*. One of them is *datasource*, which is the equivalent of *local-tx-datasource* in the AS5. The other one is *xa-datasource* being exactly the same type of *datasource* as in the AS5. The configuration of these *datasources* is similar to configuration in the AS5 with few changes. There are five configuration groups responsible for different parts of *datasource* configuration and there are:

- `<pool>` and `<xa-pool>` elements are responsible for JDBC connection pool properties where `<xa-pool>` contains few extra elements specific for the *xa-datasource*.
- `<security>` element configures connection credentials
- `<timeout>` element is for configuration of different timeouts
- `<statement>` element is for configuration of settings on statements
- `<validation>` element is for checking and validating actions done by *datasource*

A detailed description of configuration properties for both types can be found in IronJacamar<sup>4</sup> documentation and especially in its schema description.

The `<drivers>` element contains `<driver>` elements responsible for declaration of JDBC drivers that are used by *datasources*. Each `<driver>` element has a defined name by which it is referenced in *datasources* and each contains the element `<xa-datasource-class>` or `<driver-class>`. Another attribute is `module` containing information about location of *driver* library in `modules` directory.

<sup>3</sup><http://www.jboss.org/ironjacamar>

<sup>4</sup><http://www.jboss.org/ironjacamar/docs>

### 3.2 Configuration of Logging Service

Logging is a crucial component for every application and this also applies to the application server. Many administrators of application servers probably altered basic logging configuration to their specific needs. This makes logging a sensitive subject of migration. The application can help many users to migrate the logging configuration, because the AS7 is using a different logging library than AS5. This problem and its solution will be explained in this chapter.

The AS5 and earlier versions use log4j as default logging API. Log4j is very popular open source logging library from Apache and it is very similar to logging API in Java standard libraries. There are only small differences in some special features in log4j [1]. The configuration of log4j in AS5 is stored in `jboss-log4j.xml` file on the path `${JBOSS5_HOME}/server/<server>/conf/` in server file structure [1]. The configuration of log4j consists of three elements [4]:

- `<appender>` is representing output destinations for logging
- `<category>` or `<logger>` (in newer versions) is responsible for filtering logging statements by assigned levels
- `<root>` or `<root-logger>` defines parent of all loggers in hierarchy and must be always set, even if there are no other *categories*

A simplified configuration file from AS5 is shown in Listing 11.

JBoss AS 6, and more importantly the AS7, use JBoss's own implementation of logging, which is based on standard JDK logging with several upgrades [2]. However it is still possible to use log4j in application [1], but the goal of migration is to use default logging in AS7. That means it is needed to migrate configuration of log4j into AS7's logging implementation, which is described in this subsection.

Logging of AS7 can be configured in few different ways. The default way is using logging subsystem in configuration file, which acts as logging for the entire server. The AS7 configuration directory also contains the `logging.properties` file. This file is used for logging until server boots up, and the logging subsystem is still inactive. This file should not be modified and may not be in future releases of AS7 [5].

As mentioned before, both log4j and Java standard libraries logging APIs are very similar. The AS7 logging has also there main elements:

- `<handler>` has identical purpose as *appender* in log4j and similar to *appender*, there are different types *handlers*



- `<logger>` is equivalent of the *category* from log4j
- `<root-logger>` is just *root-logger* from log4j

The fresh installation of AS7 comes with basic logging configuration containing *console-handler*, *file-handler*, few *loggers* and *root-logger*. The logging subsystem is shown in Listing 12 in Appendix C.

### 3.3 Configuration of Security Service

Another important aspect of every Enterprise application is the security. The security in JBoss AS is based on Java Authentication and Authorization Service (JAAS) [2, 1] using different login modules for authentication. These modules are used by applications deployed on the server for authentication and authorization. Both versions of the application server use different approach for security built atop JAAS. Configuration of security on both versions and their migration will be described in this section.

The AS5 is using JBoss SX framework for security [1]. Main component of JBoss SX is security domain handling authorization and authentication checks of requests before they access the server. The JBoss SX is using *mbean* for configuration of security like all services in AS5 and the main functionality of this *mbean* is to load security policies. Security policies are the main configurations of security in the application server and they are defined in `jboss-login.xml` in `${JBOSSE5_HOME}/server/<server>/conf/` folder. This XML file contains different application policies. Each application policy is stored in `<application-policy>` element with defined name that can be referenced by applications. In addition, each policy contains `<authentication>` element which holds `<login-module>` elements with their specific `<module-options>`. The `<login-module>` element references *login-module* class that will be used by the domain and it also defines different types of the authorization. The `<module-option>` element defines different settings for selected *login-module* and therefore configures the security domain. Example of security configuration is shown in Listing 13 in Appendix C.

In AS7, PicketBox framework is used for security, which is qualified as extension to the application server [2]. The extension is defined in configuration file in `<extensions>` element as `<extension module="org.jboss.as.security"/>`. Configuration itself is done in security subsystem that contains different security domains in element `<security-domains>`. The `<security-domain>` elements have attribute name for referencing and can contain different `<login-module>` elements with their *module-options*, similar to AS5 configuration. Example of security subsystem is shown in Listing 14 in Appendix C

and more information can be found in AS7's documentation<sup>5</sup>.

### 3.4 Configuration of Resource Adapters

The application server is based on Java EE, which means, it can work easy with three main file types that are used in Java Enterprise applications. It is basic JAR package used for applications, WAR archive used for web applications and EAR file representing container for Enterprise applications. However there is a possibility to deploy a RAR file. The RAR file can be deployed as resource adapter in the application server. Resource adapters act as wrappers around services and they are defined by Java EE Connector Architecture (JCA) API<sup>6</sup>.

Resource adapters are defined as connection factories in AS5 and stored in files with *-ds.xml* suffix. The description of *-ds.xml* files is detailed in 3.1. The root element for connection factories in these files can contain two types of elements:

- *<tx-connection-factory>*
- *<no-tx-connection-factory>*

Similar to types of *datasources* in AS5, *tx-connection-factory* supports transactions and *no-tx-connection-factory* does not. Both resource adapters and *datasources* have common elements for the configuration with few special settings for each. In case of resource adapters, the most important elements are *<rar-name>* referencing deployed RAR file into the server, *<connection-definition>* referencing class in RAR file, *<xa-transaction>* only available in *tx-connection-factory* for declaring usage of *xa-transactions* and *<config-property>* defining property of resource adapter. The configuration and possible settings for resource adapter can be found in *jboss-ds.xsd* schema in `${JBOSS5_HOME}/docs/schema` folder and example of resource adapter in AS5 is shown in Listing 15 in Appendix C.

In AS7, the configuration of resource adapters is done in the subsystem for resource adapters. The *resource-adapter* subsystem is also provided by IronJamacar project as *datasources* subsystem and all the required information for configuration of resource adapters can be found in IronJamacar's documentation. The subsystem just contains the *<resource-adapters>* element, which contain *<resource-adapter>* elements. Resource adapters have also defined six configuration groups similar to *datasources* in the AS7, where the sixth group is *<xa-pool>* and as always they have few special settings only

<sup>5</sup><https://docs.jboss.org/author/display/AS7/Security+subsystem+configuration>

<sup>6</sup>[http://en.wikipedia.org/wiki/Java\\_EE\\_Connector\\_Architecture](http://en.wikipedia.org/wiki/Java_EE_Connector_Architecture)

for resource adapters. Example is *resource-adapter* subsystem is shown in Listing 16 in Appendix C.

### 3.5 Configuration of Web Engine

The web subsystem is intended for configuring the web engine on which web parts of enterprise applications can run. Both versions of the application server are using JBoss Web project for this job based on the popular Apache Tomcat server. Servers use different version of JBoss Web. The AS5 uses JBoss Web 2.1.x being based on Tomcat 6 and AS7 uses the newest version of JBoss Web 7 based on Tomcat 7 [2, 13]. Therefore, the migration of web engine configuration is also migration of Tomcat 6 to Tomcat 7. The configuration of web engine in both versions and its migration will be described in this section.

The basic configuration file for Tomcat is `server.xml` and this applies also to AS5. The `server.xml` file is stored in `#{JBOSS5_HOME}/server/<server>/deploy/jboss-web.sar/` folder. Information about configuration and possible settings can be found in JBoss Web 2.1.x documentation<sup>7</sup>. The configuration file has root element `<Server>` that can contain `<Service>` elements. Each `<Service>` element has a *name* attribute and element's function is to hold a group of *Connectors* associated with an *Engine*.

A `<Connector>` element has several configuration settings that can be all found in mentioned documentations. Important attribute for configuration of a *connector* is *protocol*. The *protocol* attribute defines protocol used by a *connector*. This attribute can be set only to AJP/1.3 or HTTP/1.1. The third possible protocol is HTTPS being defined by the protocol HTTP/1.1 and required SSL attributes.

Each `<Engine>` element has a *name* attribute and *default-host* attribute, which must reference a name of `<Host>` element. The `<Host>` element represents virtual server associated with network name. All the necessary information is in mentioned documentations and an example of AS5's web server configuration is shown in Listing 17 in Appendix C

The AS7 is using a centralized configuration file so there is no `server.xml` file. Instead, the configuration of web engine is done in web subsystem in configuration file. The structure is completely different and the configuration is simpler than in the AS5. The subsystem can only hold two types of elements. One is a `<connector>` element that is equivalent a *connector* in AS5 with the same functionality but with less possible settings. Another change is new *socket-binding* attribute that must reference a

<sup>7</sup><http://docs.jboss.org/jbossweb/latest/config/index.html>

`<socket-binding>` element defined in `<socket-binding-group>`. Each `<socket-binding>` element has a *name* and *port* attribute. The *name* attribute is used for referencing in configuration file and *port* attribute defines TCP port number of the socket. The second element is `<virtual-server>` containing `<alias>` elements. More information on a possible configuration can be found in AS7's documentation<sup>8</sup> and an example of web subsystem is shown in Listing 18 in Appendix C.

---

<sup>8</sup><https://docs.jboss.org/author/display/AS71/Web+subsystem+configuration>

## 4 Application for Automated Migration

Now we understand the structure of both servers and their configuration files, which is crucial for the migration. The purpose of the thesis is the automated migration of configuration of the AS5 to AS7. This chapter deals with how these configurations can be stored as objects and used in the created application for automated migration. In addition, this chapter also describes the created application and its main components and principle.

### 4.1 Lifecycle of Application

The basic idea behind an automatic migration in the application is *unmarshalling* the XML configuration files from the AS5 into objects and then creating new objects from them, representing a configuration in the AS7 and finally creating CLI commands for adding these objects into AS7 configuration. The application uses JAXB classes (*JAXB beans*) for this purpose. The process of migration and creation of new objects will be detailed in chapters describing the migration of chosen subsystems. This section is about the structure of the application and how the application works.

The application is based on the one lifecycle that migrates the configuration of AS5. The first phase of lifecycle is loading configuration files from AS5 and *unmarshalling* them into objects being afterwards stored in the migration context. The application then prepares actions created in each migrator and stores them in the migration context. This stage is the basic process of the migration of configuration because it creates actions that represent migrated configurations. Once all actions are created, the migrators are no longer needed because the application works only with actions after this point. Actions have implemented methods used by `MigratorEngine`, which controls the lifecycle of the migration.

The second phase is the pre-validation of information stored in each action. It consists of checking for example if created CLI scripts are not null or if files that should be copied into AS7 exist. If something is wrong then the action throws exception and the application stops the migration.

After a successful validation of actions each action backups necessary things. This part is mostly for actions that are responsible for copying of files because CLI commands do not need to backup anything.

Once backup is done, all actions are executed, which means different things for each type of action. Actions responsible for file transfers copy files in their destinations and, in case of modules, additional required files are also created. Actions storing CLI commands

just add their commands into the *batch* in migration context.

Then the application validates each action whether they fulfilled their purpose, which primary apply to actions responsible for file transfers and checking whether files are really in their place in the AS7 structure.

If everything went as it should then the last step is to clean created backups and to execute the *batch* with CLI commands. The execution will be done with CLI management API, which connects to the running AS7 server and executes the *batch*. If there is no problem then the migration is successful and completed, but if even one command fails then the *batch* is rolled back and the configuration of AS7 remains unchanged. The rollback is also done to other actions, which already copied files into AS7 and they are deleted or changed to their previous state.

At the moment, there are three different types of actions in the application and they are:

- `CliCommandAction` represents one CLI command and script, which will be executed on the running AS7
- `CopyAction` stores information about file that should be copied into AS7 structure to a specific destination
- `ModuleCreationAction` stores information about file that should be deployed as module in AS7

These actions are responsible for migration and each has a specific purpose. There is a possibility that new types of actions will be added for different approaches to applying the migration.

## 4.2 Structure of Created Application

The structure of the application can be seen in Figure 3. The `Ex` package contains all exceptions used in the application. The root exception is `MigratorException` extending others exceptions. This is done for better handling of exceptions if the application is used in other projects.

Another one is `Migrators` package containing five packages representing migrated subsystems by the application. More detailed description of these packages is in 4.2.2.

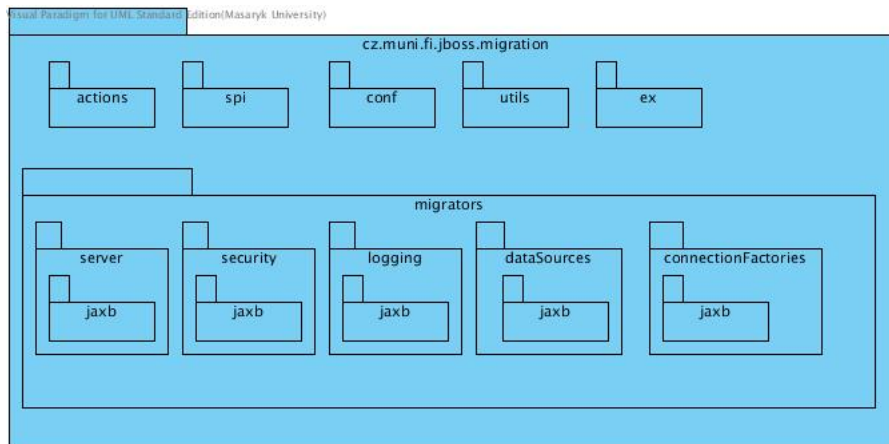


Figure 3: Structure of the Application

The `Spi` package contains all interfaces used in the application. `IConfigFragment` interface is marker that has no methods and is implemented by all AS5 JAXB beans for code readability and type safety. The `IMigrator` interface contains three methods that represent the main function of the application. Each class responsible for migration of services must implement this interface. The `IMigrator` interface is described in 4.2.1.

Another package is `utils` containing utility classes with various static methods used in other classes. These methods are usually helper methods for various purposes.

The package `conf` contains classes responsible for storing and working with given inputs and parameters for the created application.

The last package is `actions` holding classes of different actions. These actions are used in the migration.

There are several classes with different functions in the main package of the application. One of these classes is `MigrationContext` class representing context of migration keeping all the necessary information for process of migration. Another class is `MigratorEngine` responsible for process of migration of all implemented services.

#### 4.2.1 `IMigrator` Interface

The `IMigrator` interface is the backbone of the whole application. This interface is implemented by `AbstractMigrator` that is extended by all migrator classes in migrated subsystems. Its main functionality is to load the configuration from AS5 (*unmarshalling*) and creation of actions. This section describes main methods and what is their contract but first starts with `IMigrator` interface, which is shown in Listing 1.

```
public interface IMigrator {  
  
    public GlobalConfiguration getGlobalConfig();  
  
    public void setGlobalConfig(GlobalConfiguration conf);  
  
    public void loadAS5Data(MigrationContext ctx) throws LoadMigrationException;  
  
    public void createAction(MigrationContext ctx) throws MigrationException;  
  
    public int examineConfigProperty(Configuration.ModuleSpecificProperty  
                                   moduleOption);  
}
```

Listing 1: IMigrator Interface

The methods *getGlobalConfig* and *setGlobalConfig* are implemented by **AbstractMigrator** and are used for setting and getting **GlobalConfiguration** for the migration. This configuration stores all the necessary information for the application and migration itself like directories of both servers and names of the migrated profiles.

The method *loadAS5data* is used for *unmarshalling* configuration files from AS5 into objects. This method *unmarshalls* necessary files into their object representation and then stores these object into given **MigrationContext**. All migrators must override this method for their subsystem because, as it was mention before, there are different configuration files for each subsystem.

The method *createActions* represents the whole migration process. It takes objects from given **MigrationContext**, which categorizes these objects into their specific subsystem. It means, each migrator overriding this method will take only objects of its subsystem. These objects are migrated into their equivalents in AS7 configuration and method creates required actions for a successful migration.

The last method *examineConfigProperty* is a simple method for checking if given module prefix belongs to the implementation. User, for configuring the migration, defines these module options. At the moment, these options do not have any effect on migration, but they may be implemented in the future.

#### 4.2.2 Migrators Package

**Migrators** package contains five packages that represent chosen migrated subsystems in this thesis. It is possible that new subsystems for migration will be added in the future. For now, it is only web, security, logging, *datasources* and connection factories (resource adapters in AS7). Each one of them contains class or classes and another subpackage.



The subpackage is called `jaxb` and contains *JAXB beans* of both versions of server for specific subsystem represented by package. All classes are similar to the class shown in Listing 3 and they are required for a migration of each subsystem will be mentioned in chapters for that subsystem.

Each package contains at least one class responsible for manipulating with *JAXB beans* and the migration itself. Some packages can also contain some additional classes for additional functions needed in migration of specific services. Classes responsible for migration, corresponding to their package, are `ServerMigrator`, `LoggingMigrator`, `DatasourceMigrator`, `SecurityMigrator`, and `ResAdapterMigrator`. Each class extends `AbstractMigrator` and implements its abstract methods and unimplemented methods from `IMigrator` interface. In addition, each class has its own public methods for working with *unmarshalled* data and their migration.

### 4.3 JAXB and EclipseLink MOXy JAXB Implementation

We know from the previous chapter that all configuration files in the AS5 and AS7 are XML files. It is required to get all the necessary information from them that must be then migrated to their representation in the AS7. The solution for an easy manipulation with XML content is Java object oriented approach using JAXB.

Java Architecture for XML Binding (JAXB) is a specification that allows mapping between XML documents and Java objects. The JAXB framework provides two main functionalities:

- Unmarshalling – retrieving XML content into Java object representation
- Marshalling – putting Java object representation into XML document

All information in this section about JAXB can be found on the JAXB project documentation site [10].

It is first required to define Java classes to represent XML documents. These classes are regular classes with JAXB annotations that will represent the content of XML document as objects. JAXB annotations define which attribute represents which information from XML. There are three basic JAXB annotations, which are used frequently in almost every JAXB class:

- `@XmlElement` defines the root element of the object and it is also required for classes representing children elements of the root element, not only on classes representing the root element of the XML document

- `@XmlAttribute` defines which attributes of the class are representing attributes of the root element
- `@XmlElement` also defines, which attributes of a class are representing children elements
- `@XmlElements` has similar functionality as `@XmlElement` for multiple elements

Using these annotations can be useful in XML documents of a simple structure. XML documents with many nested elements require more than one class. This problem can be avoided using XPath. Unfortunately, this option is not available in basic JAXB specification so it is required to use some extension.

EclipseLink MOXy is a JAXB implementation with additional features that can help with mapping of complex XML structures [11]. The most important new feature for us is the `@XPath` annotation allowing XPath expression. This helps to reduce the number of required Java classes for XML documents and is very useful in many situations.

All Java classes in the application used for *marshalling* and *unmarshalling* were created manually. JAXB also gives options to generate classes from given XML schema, but this process was not used because it produces a lot of classes, which makes working with them difficult. Manually created classes are better for maintaining or possible upgrades.

Example of an XML document with a created JAXB class from the migration application can be seen in Listing 2 and Listing 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<appender name="FILE" class="DailyRollingFileAppender">
  <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
  <param name="File" value="{jboss.server.log.dir}/server.log"/>
  <param name="Append" value="true"/>
  <param name="DatePattern" value="'. 'yyyy-MM-dd"/>
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d%-5p[%c](%t)%m%n"/>
  </layout>
</appender>
```

Listing 2: Log4j Appender Example

The attribute *parameters* in Listing 3 has a special annotation for multiple occurrences of one element and the type of the attribute is class `ParameterBean`. This is an example of a problem when working with JAXB. It has a problem with multiple occurrences of element with two or more attributes. The only solution is to define another class for that element. `ParameterBean` is exactly this type of class with only two attributes with JAXB annotations shown in Listing 4.

```
@XmlElement(name = "appender")
public class AppenderBean{
    @XmlAttribute(name = "name")
    private String appenderName;

    @XmlAttribute(name = "class")
    private String appenderClass;

    @XmlElement(@XmlElement(name = "param", type = ParameterBean.class))
    private Set<ParameterBean> parameters;

    @XPath("appender-ref/@ref")
    private Set<String> appenderRefs;

    @XPath("layout/param/@name")
    private String layoutParamName;

    @XPath("layout/param/@value")
    private String layoutParamValue;

    // Setters and getters methods are omitted in this example!
}
```

Listing 3: Class with JAXB Annotations

```
@XmlElement(name = "parameter")
public class ParameterBean {

    @XmlAttribute(name = "name")
    private String paramName;

    @XmlAttribute(name = "value")
    private String paramValue;

    // Setters and getters methods are omitted in this example!
}
```

Listing 4: ParameterBean Class

## 5 Implementation of Migration

Using all information about style and possible options of configuration of each service from Chapter 3, there is a way to migrate and implement this migration of configuration from AS5 to AS7. This implementation for each service is detailed in this chapter along with solutions to various problems arising from the changed configuration.

The migration in general starts with *unmarshalling* all configuration files from AS5 to objects and storing them into the migration context. Objects are stored in a map that sorts them by subsystems. Then each migrator tries to migrate these objects to their equivalents in AS7 configuration and creates from these migrated objects actions that are stored into the migration context. These actions are then executed from migration context and the migration is done.

### 5.1 Migration of Datasource Configuration

The migration of *datasources* is really straightforward with only few small problems but the full migration of *datasource* is more difficult because it is required to migrate JDBC driver into the AS7 as module.

As we know, there are three types of *datasources* in the AS5 and only two in the AS7, so this is the first problem of the migration. From the description of the configuration on both application servers, it is clear that there is no *no-tx-datasource* type of *datasource* in the AS7. The solution of this problem is to create a basic *datasource* with a special attribute *jta* set to false, which will disable JTA transactions.

*Datasources* in the AS7 have almost all possible settings from the AS5 with few changes in names and few new ones added in the AS7 configuration. Configuration groups are one of them, which means, most elements have just changed their position in XML structure. There is also added one new required attribute *pool-name*. This attribute is equivalent of `<jndi-name>` element in AS5 but AS7's configuration has also attribute *jndi-name*. The *jndi-name* attribute in AS7 must have special prefix *java:jboss/datasource/*, that usually ends by *pool-name*. The whole migration is very clear just from example of AS5's *datasource* and its AS7 equivalent shown in Appendix C in Listing 8 and Listing 9.

The more important and more difficult part of the *datasource* migration is the migration of *driver* and *driver's* library into AS7. Two of the elements not appearing in *datasource* configuration in the AS7 are `<driver-class>` and `<xa-datasource-class>`. As mentioned in 3.1, they must be declared in separate element `<driver>` in element `<drivers>`. Each `<driver>` element must have a defined name by which it can be ref-

erenced in *datasources*. The name of the *driver* is set to *createdDriverX* where X is the number of migrated driver.

This process of migration is successful but it is still incomplete because the *driver* also needs a *driver's* library containing the mentioned class. Without this library deployed in the AS7, the *driver* is unusable and likewise *datasources* that are using it. The deployment of the library is done by the application that will add *driver's* library into the AS7 instead of the user.

The application tries first to find the declared *driver's* class in JAR archives in `${JBOSSE5_HOME}/commons/lib` or `${JBOSSE5_HOME}/server/<server-name>/lib` directories. Then it must create module for this file, starting with creation of folder structure `${JBOSSE7_HOME}/modules/migration/drivers`. For each migrated *driver* from the AS5, it creates subfolder *createdDriverX* where X is the number of migrated driver, and it must contain subfolder *main*. The application then copies found library into the *main* folder and then creates `module.xml` file in it.

The `module.xml` file is necessary for importing modules into the AS7. It defines name of the module, which can be then used in the configuration file. It also defines which libraries in folder are used when the module is called and module's dependencies on other modules. In our situation, there are required three main dependencies for *driver* and there are *javax.api*, *javax.transaction.api* and *javax.servlet.api*. Last thing to do is to reference the created module in `<driver>` element. An example of created *driver* is shown in Listing 6 and an example of `module.xml` is shown in Listing 5.

```
<module xmlns="urn:jboss:module:1.1" name="jdbc.drivers.mssql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.22-bin.jar"/>
    <!-- Insert resources here -->
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.servlet.api" optional="true"/>
  </dependencies>
</module>
```

Listing 5: Example of `module.xml`

```
<driver name="mssql" module="jdbc.drivers.mssql">
  <driver-class>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-class>
</driver>
```

Listing 6: Example of Driver Element

## 5.2 Migration of Logging Configuration

It is clear from previous examples of configurations on both versions of AS that migration itself is not so difficult. Migration itself is break down to migrating *appenders* to *handlers*, *categories* to *loggers* and changing syntax of the *root-logger* from AS5 to AS7.

Start with the migration of *category* and *root-logger* is really straightforward. The main change is of course the name of elements. Changes are also made to two children elements in both. The first element is `<priority>` that changes name to `<level>`. The last element `<appender-ref>` needs to be changed to element `<handler>` and declared in parent element `<handlers>`. There can be more than one element of this type in both elements. An example of *category* migration is shown in Listing 7.

```
<category name="org.jboss.ha">
  <priority value="DEBUG" />
  <appender-ref ref="CLUSTER"/>
</category>

<logger category="org.jboss.ha">
  <level name="WARN"/>
  <handlers>
    <handler name="CLUSTER"/>
  </handlers>
</logger>
```

Listing 7: Example of Category Migration

As it was mentioned before, log4j has only one element for all *appenders* in configuration. The type of *appender* is stored in attribute *class*. There are few common types in log4 itself, but there is also an option to define and use your own class for logging. On the other hand, logging in AS7 has unique elements and attributes for all supported handlers. That means, the first step is to parse value stored in *appender's* attribute *class* and finding the right type of *handler* in AS7. This value is in Java naming convention, so that means, it contains full package path ending with name of the class and possibilities are:

- *DailyRollingFileAppender* will be migrated to `<periodic-rotating-file-handler>`
- *RollingFileAppender* will be migrated to `<size-rotating-file-handler>`
- *ConsoleAppender* will be migrated to `<console-handler>`
- *AsyncAppender* will be migrated to `<async-handler>`
- Special types of *appenders* will be migrated to `<custom-handler>`.

The next step is to migrate all settings of *appenders*. These settings are represented by element `<param>`, which has attributes *name* and *value*. This step is basically identical to the first one. The application parses attribute *name* and finds the corresponding setting in AS7's configuration and stores in it the attribute *value*. The right syntax of settings for all *handlers* can be found in AS7's documentation [5].

However, special *appenders* migration is more complicated. *Appendes* not having representation in specific *handler* must be migrated to *custom-handlers*. These special *appenders* either use special log4j classes or classes created by users for their specific logging. Classes created by users must be first found in deployed JAR files in AS5 file structure. These files can be found in `${JBOSSE5_HOME}/common/lib` or `${JBOSSE5_HOME}/server/<server-name>/lib` folders. They are then copied to created module in `modules` directory in AS7 in subfolder `migration/logging/customHandlerX`. Similar to creation of *driver* module, `module.xml` must be created for new modules with different dependencies than in drivers. The log4j module already exists in AS7, so no copying is required. A *custom-handler* then references migrated class along with created module or log4j module in AS7.

### 5.3 Migration of Security Configuration

The migration itself is not difficult because each `<application-policy>` is just changed to `<security-domain>` but there is change required in `<login-module>` and `<module-options>` elements. The `<login-module>` element references the *login-module* class with full package name but AS7 configuration is using only the name of the class, in other words, only type of *login-module*. The application parses the class in AS5 configuration and chooses the adequate type for AS7 configuration. All possible types of *login-modules* are listed in the documentation of the security subsystem mentioned in 3.3.

The last change is in the `<module-option>` element where text content in AS5 must be moved to attribute *value*. The *module-options* can also contain reference on properties files used for security that must be copied to AS7. The application copies these files in `${JBOSSE7_HOME}/standalone/configuration` folder and changes their relative path in AS7 configuration to their new location. An example of the migration is shown in Appendix C in Listing 13 and Listing 14.

### 5.4 Migration of Resource Adapters Configuration

The migration of resource adapters is relatively easy and very similar to migration of *datasources*. As in *datasources* many elements just have slightly changed name and

position in XML but, once again, it is necessary to deploy referenced RAR file into AS7. The basic migration can be understood just from example of AS5 configuration with its migrated equivalent. Both examples are shown in Listing 15 and Listing 16 in Appendix C. Problematic parts and their solution will be described in this section.

The first problem arises from the two types of adapters in AS5 and only one available type in AS7. At first glance it seems that AS7 no longer supports adapters with different transaction support but it is not true. The resource adapter in AS7 has special configuration element `<transaction-support>` with settings `NoTransaction` used for adapter without transaction support, `LocalTransaction` for adapter with transaction support and `XaTransaction` for adapter with xa-transaction support. The xa-transaction support is defined by children element `<xa-transaction>` in `<tx-connection-factory>` in AS5. Another problem is with attributes `pool-name` and `jndi-name`. It is exactly the same problem as in the migration of `datasources` but the prefix for `jndi-name` must be `java:jboss/` in this situation.

Last thing is a deployment of the RAR file into AS7 that is easier than the deployment of `driver` because it does not need to be deployed as module. The application must first find referenced file in AS5 structure and then it must be copied into `#{JBOSS5_HOME}/standalone/deployments` folder. After that the file is successfully deployed into server but there is a possibility that the deployed RAR file will need some alternation by user for non-problematic deployment.

## 5.5 Migration of Web Engine Configuration

Migration of previous subsystems was relatively easy because there were similarities in configurations but a web engine configuration is completely different. One of the key features of the new configuration is simplicity and, because of that, it has less configuration possibilities than AS5 configuration and the completely different structure. The first thing to consider is that it no more associates group of `connectors` with `engines` or `virtual servers`, in other words to say. The AS7 configuration also has no `<Realm>` element, so this element is completely ignored in process of migration. The `<Valve>` element is added to the configuration of web subsystem in JBoss AS 7.2.x but the final version released does not support this element at the moment. It can instead be defined and configured in deployed applications. Currently, the application ignores this element in process of migration and only `Connectors` and `Engine` is migrated in each `Service`.

`Connectors` in AS7 have many similar attributes as in AS5 configuration but few settings were deleted and there are few changes in structure. The first one is that



configuration of SSL is now done in `<ssl>` children element and settings associated with SSL are attributes of this element. The SSL configuration also lost few settings and some of them were merged to one. The list of all possible settings of connector in both versions is very long and describing migration or change of each attribute would take long. Lot of information can be found in mentioned documentations or in code in `ServerMigrator`.

Another change is declaration of port number in `<connector>` element that must be done as `<socket-binding>` element. The AS7 come with some default `socket-bindings` like default ports for AJP, HTTP and HTTPS protocols. The application parses `socket-bindings` in given configuration file and set `socket-binding` attribute of migrated `connector` to one of them if it is already defined. If `connector` is using port that is not defined as `<socket-binding>`, the application creates new `<socket-binding>` for the port number. This creates problem because one `<socket-binding>` can be used only by one `connector`. In the situation when this problem occurs, it is up to the user to solve it manually after the migration.

The `<Engine>` element is migrated to a `<virtual-server>` element with the name of `Engine` and each `<Host>` element is changed to `<Alias>` element. Each `<Host>` element can hold number of `<Alias>` elements. These elements are also migrated to `<alias>` elements in `<virtual-server>`. Only `name` attributes are migrated because other settings in both elements are no longer used in AS7 configuration. An example of migration to web subsystem in AS7 is shown in Listing 17 and Listing 18 in Appendix C.

## 6 Conclusion and Use of Application

### 6.1 Using the Created Application

To use the application, we need to start the JBoss AS7 in admin-only mode. This mode is ideal for configuring the application server because it starts the server without additional services, which means, the application can add problematic parts of the migration into the configuration. Starting the application server in admin-only mode is shown in Figure 4.

```
[22:44:05 /home]>jboss-as-7.1.1.Final/bin/standalone.sh --admin-only
=====
JBoss Bootstrap Environment

JBOSS_HOME: /home/jboss-as-7.1.1.Final

JAVA: /usr/local/java/jdk1.7.0_21/bin/java

JAVA_OPTS: -server -XX:+UseCompressedOops -XX:+TieredCompilation -Xms64m -Xmx512m -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true -Djboss.server.default.config=standalone.xml
=====

22:44:08,553 INFO [org.jboss.modules] JBoss Modules version 1.1.1.GA
22:44:08,757 INFO [org.jboss.msc] JBoss MSC version 1.0.2.GA
22:44:08,809 INFO [org.jboss.as] JBAS015899: JBoss AS 7.1.1.Final "Brontes" starting
22:44:09,622 INFO [org.xnio] XNIO Version 3.0.3.GA
22:44:09,622 INFO [org.jboss.as.server] JBAS015888: Creating http management service using socket-binding (management-http)
22:44:09,636 INFO [org.xnio.nio] XNIO NIO Implementation Version 3.0.3.GA
22:44:09,649 INFO [org.jboss.remoting] JBoss Remoting version 3.2.3.GA
22:44:09,704 INFO [org.jboss.as.remoting] JBAS017100: Listening on /127.0.0.1:9999
22:44:12,295 INFO [org.jboss.as] JBAS015951: Admin console listening on http://127.0.0.1:9990
22:44:12,296 INFO [org.jboss.as] JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 4023ms - Started 26 of 45 services (19 services are passive or on-demand)
```

Figure 4: Starting the Application Server in admin-only Mode

With the application server running in admin-only mode we can use the created application. The application is packed in JAR archive, which can be executed from the command line with parameters. There are two required parameters for setting directories of running JBoss AS 7 and to be migrated JBoss AS 5, there are also other optional settings. All possible settings for the application are shown in Appendix B. Starting the application with only required parameters results in default settings for profiles used in the migration for both versions of the application server. Use of the application and successful run is shown in Figure 5.

From Figure 5, we can see which actions were created by the application, it has also connected to the application server and executed actions representing CLI command as batch on it. By checking the configuration file of the AS7, default file is set to `standalone.xml` in this case; we will see that the configuration file is altered and the configuration of AS5 was migrated along with required files found in the AS7 structure. The last step is to shutdown and start the application server without admin-only mode and check whether all services are running as they should and if the application server



process. All problems mentioned in the Chapter 5 are also shown in the examples with exceptions to examples of created modules. These examples provide the basic assumption that migration is successful but the only way how to assure that migration is working is starting AS7 and check whether all services are working as they should on the installed application server. Without using the application, it is also required to deploy files referenced in the examples or otherwise migrated services would not work. Taking everything into consideration, we can say that thesis is describing migration of few chosen services and sets basics for migration of other services. That means that the first goal of this thesis set in the beginning has been fulfilled.

The application attached to the thesis is the result and the implementation of the theory behind the migration. It is also capable of automated migration along with deploying referenced files in migrated configurations in they right place in AS7 structure. That means the application fulfills the second goal of this thesis. There is only one problem with automated migration at the moment. This problem is that each service has few specific settings that required user manipulation. This problem mainly consists of integration of created configuration with the default configuration set in fresh install of AS7. The application generates CLI commands and scripts that provide validation of settings declared in scripts and can help users to solve problems if they arise. All CLI scripts created by the application for all examples in Appendix C are show in Appendix D.

Given these problems, it is really hard and almost impossible to create application capable of fully automated migration of JBoss AS5 without user interaction. These problems can be also solved by giving users a certain way to choose and manipulate what should be migrated and what should not be. These settings can be set in the application as its parameters when user executes application JAR file. At the moment, the application saves these settings but they do not have affect of migration. I hope, this feature will be implemented in the future because there is already prepared ground for this feature. That means the application also meets the third goal of this thesis because it was developed with intention for easy expansion of functionality and further development in the open-source community.

Red Hat is interested in this idea and especially in the application created for this thesis and its further development in the open source community. This will provide significant upgrades and new features for the application like new migration of other services and more importantly migration of cluster configuration, which is used the most by customers. Also there is a possibility for a better support of user input or even GUI for user's interaction. In addition, there is also a possibility for integration with other projects focusing on migration of application like JBoss Cake, which can become

the universal tool for migration in all fields. The version of the created application that is packed with this thesis is covering all services covered in this thesis and API for it is displayed in Appendix B.

# Appendices

## A Attached files

There are three files attached in the electronical submission of this thesis:

- *AsMigration-1.0.jar* is created application for this thesis that is already build into JAR archive and can be executed from command line
- *AsMigrationSource.zip* contains source code snapshot of the created application that is buildable by Maven
- *Examples.zip* contains XML files shown in Appendix C along with default configuration file for JBoss AS 7. It also contains configuration file of JBoss AS 7 with applied migration of configuration and created CLI scripts shown in Appendix D.

## B Usage of Application

### Usage:

- `java -jar AsMigration-1.0.jar -as5.dir=<AS5path> -as7.dir=<AS7.path> [<option>, ...]`

### Options:

- `-as5.profile=<name>`  
Path to AS 5 profile. Default: "default"
- `-as7.confPath=<path>`  
Path to AS 7 config file. Default: "standalone/configuration/standalone.xml"
- `-conf.<module>.<property>=<value>` := Module-specific options.  
`<module>` := Name of one of modules. E.g. datasource, security, ...  
`<property>` := Name of the property to set. Specific per module. May occur multiple times.  
-> \*This option doesn't have any effect at this moment. May be implemented in the future\*

## C Configuration Examples

```
<datasources>
  <no-tx-datasource>
    <jndi-name>MediaWikiDBImporterRoleDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/MediaWikiDB</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>username</user-name>
    <password>password</password>
  </no-tx-datasource>
  <local-tx-datasource>
    <connection-property name="char.encoding">UTF-8</connection-property>
    <connection-property name="test">testing</connection-property>
    <jndi-name>DefaultDS5</jndi-name>
    <connection-url>jdbc:mysql://localhost:3307/localDB</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>sa</user-name>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>1</idle-timeout-minutes>
    <security-domain>HsqlDbRealm</security-domain>
    <prepared-statement-cache-size>32</prepared-statement-cache-size>
  </local-tx-datasource>
  <xa-datasource>
    <jndi-name>jos</jndi-name>
    <xa-datasource-class>
      com.mysql.jdbc.jdbc2.optional.MySQLXADataSource
    </xa-datasource-class>
    <xa-datasource-property name="URL">http://testURL.com</xa-datasource-property>
    <user-name>testUser</user-name>
    <password>password</password>
    <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>100</max-pool-size>
    <blocking-timeout-millis>2000</blocking-timeout-millis>
    <idle-timeout-minutes>2</idle-timeout-minutes>
    <track-connection-by-tx />
    <valid-connection-checker-class-name>
      org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker
    </valid-connection-checker-class-name>
    <exception-sorter-class-name>
      org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter
    </exception-sorter-class-name>
    <metadata>
      <type-mapping>mySQL</type-mapping>
    </metadata>
  </xa-datasource>
</datasources>
```

Listing 8: Example of Datasource Configuration in JBoss AS 5



```

<subsystem xmlns="urn:jboss:domain:datasources:1.0">
  <datasources>
    <datasource jndi-name="java:jboss/datasources/DefaultDS5" pool-name="DefaultDS5">
      <connection-url>jdbc:mysql://localhost:3307/localDB</connection-url>
      <driver>createdDriver1</driver>
      <pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>20</max-pool-size>
      </pool>
      <security>
        <user-name>sa</user-name>
        <security-domain>HsqlDbRealm</security-domain>
      </security>
      <timeout>
        <idle-timeout-minutes>1</idle-timeout-minutes>
      </timeout>
    </datasource>
    <datasource jta="false" jndi-name="java:jboss/datasources/MediaWikiDBImporterRoleDS"
      pool-name="MediaWikiDBImporterRoleDS">
      <connection-url>jdbc:mysql://localhost:3306/MediaWikiDB</connection-url>
      <driver>createdDriver1</driver>
      <security>
        <user-name>username</user-name>
        <password>password</password>
      </security>
    </datasource>
    <xa-datasource jndi-name="java:jboss/datasources/jos" pool-name="jos">
      <xa-datasource-property name="URL">http://testURL.com</xa-datasource-property>
      <driver>createdDriver2</driver>
      <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
      <xa-pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>100</max-pool-size>
      </xa-pool>
      <security>
        <user-name>testUser</user-name>
        <password>password</password>
      </security>
      <validation>
        <exception-sorter
          class-name="org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter"/>
      </validation>
      <timeout>
        <idle-timeout-minutes>2</idle-timeout-minutes>
      </timeout>
      <drivers/>
    </xa-datasource>
  </datasources>
</subsystem>

```

Listing 9: Migrated Configuration of Datasources

```
<drivers>
  <driver name="createdDriver2" module="migration.drivers.createdDriver2">
    <xa-datasource-class>c
      com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
    </xa-datasource-class>
  </driver>
  <driver name="createdDriver1" module="migration.drivers.createdDriver2">
    <driver-class>com.mysql.jdbc.Driver</driver-class>
  </driver>
</drivers>
```

Listing 10: Drivers Element from Datasource Subsystem

```

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">
  <appender name="FileAppender"
    class="org.jboss.logging.appender.DailyRollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="{jboss.server.log.dir}/server.log"/>
    <param name="Append" value="true"/>
    <param name="DatePattern" value="',.'yyyy-MM-dd"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d%-5p[%c](%t)%m%n"/>
    </layout>
  </appender>
  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="Target" value="System.out"/>
    <param name="Threshold" value="INFO"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{ABSOLUTE}%-5p[%c{1}]%m%n"/>
    </layout>
  </appender>
  <category name="org.apache">
    <priority value="INFO"/>
  </category>
  <category name="org.jgroups">
    <priority value="WARN"/>
  </category>
  <category name="org.quartz">
    <priority value="INFO"/>
    <priority appender-ref="FileAppender"/>
    <priority appender-ref="ConsoleAppender"/>
  </category>
  <root>
    <priority value="DEBUG"/>
    <appender-ref ref="ConsoleAppender"/>
    <appender-ref ref="FileAppender"/>
  </root>
</log4j:configuration>

```

Listing 11: Example of Logging Configuration in JBoss AS 5

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">
  <console-handler name="ConsoleAppender">
    <level name="INFO"/>
    <formatter>
      <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c{1}] %m%n"/>
    </formatter>
    <target name="System.out"/>
  </console-handler>
  <periodic-rotating-file-handler name="FileAppender">
    <formatter>
      <pattern-formatter pattern="%d %-5p [%c] (%t) %m%n"/>
    </formatter>
    <file relative-to="jboss.server.log.dir" path="server.log"/>
    <suffix value="'. 'yyyy-MM-dd"/>
    <append value="true"/>
  </periodic-rotating-file-handler>
  <logger category="org.jgroups">
    <level name="WARN"/>
  </logger>
  <logger category="org.apache">
    <level name="INFO"/>
  </logger>
  <logger category="org.quartz">
    <level name="INFO"/>
    <handlers>
      <handler name="FileAppender"/>
      <handler name="ConsoleAppender"/>
    </handlers>
  </logger>
  <root-logger>
    <level name="DEBUG"/>
    <handlers>
      <handler name="FileAppender"/>
      <handler name="ConsoleAppender"/>
    </handlers>
  </root-logger>
</subsystem>
```

Listing 12: Migrated Logging Configuration

```
<policy>
  <application-policy name="client-login">
    <authentication>
      <login-module code="org.jboss.security.ClientLoginModule"
        flag="required">
        <module-option name="restore-login-identity">true</module-option>
      </login-module>
    </authentication>
  </application-policy>
  <application-policy name="HsqlDbRealm">
    <authentication>
      <login-module code="org.jboss.resource.security.ConfiguredIdentityLoginModule"
        flag="required">
        <module-option name="principal">sa</module-option>
        <module-option name="userName">sa</module-option>
        <module-option name="password">sa</module-option>
      </login-module>
    </authentication>
  </application-policy>
  <application-policy name="JBossWS">
    <authentication>
      <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
        flag="required">
        <module-option name="usersProperties">
          props/jbossws-users.properties
        </module-option>
        <module-option name="rolesProperties">
          props/jbossws-roles.properties
        </module-option>
        <module-option name="unauthenticatedIdentity">anonymous</module-option>
      </login-module>
    </authentication>
  </application-policy>
</policy>
```

Listing 13: Example of Security Configuration in JBoss AS 5

```
<subsystem xmlns="urn:jboss:domain:security:1.1">
  <security-domains>
    <security-domain name="client-login">
      <authentication>
        <login-module code="Client" flag="required">
          <module-option name="restore-login-identity" value="true"/>
        </login-module>
      </authentication>
    </security-domain>
    <security-domain name="HsqlDbRealm">
      <authentication>
        <login-module code="ConfiguredIdentity" flag="required">
          <module-option name="userName" value="sa"/>
          <module-option name="password" value="sa"/>
          <module-option name="principal" value="sa"/>
        </login-module>
      </authentication>
    </security-domain>
    <security-domain name="JBossWS">
      <authentication>
        <login-module code="UsersRoles" flag="required">
          <module-option name="unauthenticatedIdentity" value="anonymous"/>
          <module-option name="usersProperties"
            value="${jboss.server.config.dir}/jbossws-users.properties"/>
          <module-option name="rolesProperties"
            value="${jboss.server.config.dir}/jbossws-roles.properties"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

Listing 14: Migrated Security Configuration

```
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>XUConnectionFactory</jndi-name>
    <rar-name>genericra.rar</rar-name>
    <connection-definition>
      com.sun.genericra.outbound.ManagedQueueConnectionFactory
    </connection-definition>
    <adapter-display-name>XU</adapter-display-name>
    <max-pool-size>50</max-pool-size>
    <blocking-timeout-millis>50000</blocking-timeout-millis>
    <idle-timeout-minutes>15</idle-timeout-minutes>
  </no-tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>genericra.rar</rar-name>
    <connection-definition>
      com.sun.genericra.outbound.ManagedJMSConnectionFactory
    </connection-definition>
    <config-property name="ConnectionValidationEnabled" type="java.lang.Boolean">
      true
    </config-property>
    <min-pool-size>10</min-pool-size>
    <max-pool-size>10</max-pool-size>
    <username>testUser</username>
    <password>pass</password>
    <background-validation>false</background-validation>
    <background-validation-millis>5000</background-validation-millis>
    <blocking-timeout-millis>4000</blocking-timeout-millis>
    <idle-timeout-minutes>2</idle-timeout-minutes>
    <allocation-retry>5</allocation-retry>
    <allocation-retry-wait-millis>4000</allocation-retry-wait-millis>
  </tx-connection-factory>
</connection-factories>
```

Listing 15: Example of Resource Adapters Configuration in JBoss AS 5

```

<subsystem xmlns="urn:jboss:domain:resource-adapters:1.0">
  <resource-adapters>
    <resource-adapter>
      <archive>genericra.rar</archive>
      <transaction-support>XATransaction</transaction-support>
      <connection-definitions>
        <connection-definition>
          class-name="com.sun.genericra.outbound.ManagedJMSConnectionFactory"
          jndi-name="java:jboss/JBossTestCF2" use-java-context="true"
          pool-name="JBossTestCF2" enabled="true">
            <config-property name="ConnectionValidationEnabled">
              true
            </config-property>
            <xa-pool>
              <min-pool-size>10</min-pool-size>
              <max-pool-size>10</max-pool-size>
            </xa-pool>
            <timeout>
              <idle-timeout-minutes>2</idle-timeout-minutes>
              <allocation-retry>5</allocation-retry>
              <allocation-retry-wait-millis>4000</allocation-retry-wait-millis>
            </timeout>
            <validation>
              <background-validation>>false</background-validation>
              <background-validation-millis>5000</background-validation-millis>
            </validation>
          </connection-definition>
        </connection-definitions>
      </resource-adapter>
    <resource-adapter>
      <archive>genericra.rar</archive>
      <transaction-support>NoTransaction</transaction-support>
      <connection-definitions>
        <connection-definition>
          class-name="com.sun.genericra.outbound.ManagedQueueConnectionFactory"
          jndi-name="java:jboss/XUConnectionFactory" use-java-context="true"
          pool-name="XUConnectionFactory" enabled="true">
            <pool>
              <max-pool-size>50</max-pool-size>
            </pool>
            <timeout>
              <idle-timeout-minutes>15</idle-timeout-minutes>
            </timeout>
          </connection-definition>
        </connection-definitions>
      </resource-adapter>
    </resource-adapters>
  </subsystem>

```

Listing 16: Migrated Resource Adapters Configuration



```
<Server>
  <Listener className="org.apache.catalina.core.AprLifecycleListener"
    SSLEngine="on"/>
  <Listener className="org.apache.catalina.core.JasperListener"/>
  <Service name="jboss.web">
    <Connector protocol="HTTP/1.1" port="8000" address="{jboss.bind.address}"
      connectionTimeout="20000" redirectPort="8443" />
    <Connector protocol="AJP/1.3" port="8009" address="{jboss.bind.address}"
      redirectPort="8443" />
    <Connector protocol="HTTP/1.1" SSLEnabled="true"
      port="8449" address="{jboss.bind.address}"
      scheme="https" secure="true" clientAuth="false"
      keystoreFile="{jboss.server.home.dir}/conf/chap8.keystore"
      keystorePass="rmi+ssl" sslProtocol = "TLS" />
    <Engine name="jboss.web" defaultHost="localhost">
      <Realm className="org.jboss.web.tomcat.security.JBossWebRealm"
        certificatePrincipal="org.jboss.security.auth.certs.SubjectDNMapping"
        allRolesMode="authOnly"/>
      <Host name="localhost">
        <Alias name="company.com"/>
        <Valve className="org.jboss.web.tomcat.service.jca.CachedConnectionValve"
          transactionManagerObjectName="jboss:service=TransactionManager" />
      </Host>
      <Host name="example.com">
        <Alias name="example.org"/>
      </Host>
    </Engine>
  </Service>
</Server>
```

Listing 17: Example of Web Engine Configuration in JBoss AS 5

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-server="default-host"
  native="false">
  <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http"/>
  <connector name="connector1" protocol="HTTP/1.1" scheme="http"
    socket-binding="createdSocket" redirect-port="8447"/>
  <connector name="connector2" protocol="AJP/1.3" socket-binding="ajp"
    redirect-port="8445" scheme="http"/>
  <connector name="connector3" protocol="HTTP/1.1" scheme="https" secure="true"
    socket-binding="https">
    <ssl name="ssl" password="changeit"
      certificate-key-file="{jboss.server.config.dir}/keys/chap8.keystore"
      protocol="TLS" verify-client="false"/>
  </connector>
  <virtual-server name="default-host" enable-welcome-root="true">
    <alias name="localhost"/>
    <alias name="example.com"/>
  </virtual-server>
  <virtual-server name="jboss.web" enable-welcome-root="true">
    <alias name="company.com"/>
    <alias name="example.org"/>
    <alias name="localhost"/>
    <alias name="example.com"/>
  </virtual-server>
</subsystem>
```

Listing 18: Migrated Web Engine Configuration

## D Examples of Created CLI Scripts

```
/subsystem=datasources/jdbc-driver=createdDriver2:add(  
driver-module-name=migration.drivers.createdDriver2,  
driver-xa-datasource-class-name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource)  
  
/subsystem=datasources/jdbc-driver=createdDriver1:add(  
driver-module-name=migration.drivers.createdDriver2,  
driver-class-name=com.mysql.jdbc.Driver)  
  
data-source add name=DefaultDS5 jndi-name=java:jboss/datasources/DefaultDS5  
driver-name=createdDriver1 connection-url=jdbc:mysql://localhost:3307/localDB  
min-pool-size=5 max-pool-size=20 user-name=sa security-domain=HsqlDbRealm  
idle-timeout-minutes=1  
  
xa-data-source add name=jos jndi-name=java:jboss/datasources/jos min-pool-size=5  
driver-name=createdDriver2 transaction-isolation=TRANSACTION_READ_COMMITTED  
max-pool-size=100 password=password user-name=testUser idle-timeout-minutes=2  
exception-sorter-class-name=org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter  
blocking-timeout-millis=2000  
  
/subsystem=datasources/xa-data-source=jos/xa-datasource-properties=URL:add(  
value=http://testURL.com)  
  
data-source add name=MediaWikiDBImporterRoleDS  
jndi-name=java:jboss/datasources/MediaWikiDBImporterRoleDS jta=false  
driver-name=createdDriver1 password=password user-name=username  
connection-url=jdbc:mysql://localhost:3306/MediaWikiDB
```

Listing 19: CLI Scripts for Adding Migrated Datasources

```

/subsystem=logging/logger=org.quartz:add(handlers=["FileAppender","ConsoleAppender"],
level=INFO)

/subsystem=logging/logger=org.jgroups:add(level=WARN)

/subsystem=logging/logger=org.apache:add(level=INFO)

/subsystem=logging/periodic-rotating-file-handler=FileAppender:add(append=true,
file={"relative-to"=>"jboss.server.log.dir", "path"=>"server.log"},
suffix='.yyyy-MM-dd', formatter=%d %-5p [%c] (%t) %m%n)

/subsystem=logging/console-handler=ConsoleAppender:add(level=INFO, target=System.out,
formatter=%d{HH:mm:ss,SSS} %-5p [%c{1}] %m%n)

/subsystem=logging/root-logger=ROOT:write-attribute(name=level, value=DEBUG)

/subsystem=logging/root-logger=ROOT:write-attribute(name=handlers, handlers=
["FileAppender","ConsoleAppender"])

```

Listing 20: CLI Scripts for Adding the Migrated Logging Configuration

```

/subsystem=security/security-domain=JBossWS:add(cache-type=default)

/subsystem=security/security-domain=JBossWS/authentication=classic:add(
login-modules=[{"code"=>"UsersRoles", "flag"=>"required",
"module-option"=>[("unauthenticatedIdentity"=>"anonymous"),
("rolesProperties"=>"${jboss.server.config.dir}/jbossws-roles.properties"),
("usersProperties"=>"${jboss.server.config.dir}/jbossws-users.properties")]

/subsystem=security/security-domain=HsqlDbRealm:add(cache-type=default)

/subsystem=security/security-domain=HsqlDbRealm/authentication=classic:add(
login-modules=[{"code"=>"ConfiguredIdentity", "flag"=>"required",
"module-option"=>[("principal"=>"sa"), ("userName"=>"sa"), ("password"=>"sa")]

/subsystem=security/security-domain=client-login:add(cache-type=default)

/subsystem=security/security-domain=client-login/authentication=classic:add(
login-modules=[{"code"=>"Client", "flag"=>"required",
"module-option"=>[("restore-login-identity"=>"true")]

```

Listing 21: CLI Scripts for Adding the the Migrated Security Configuration

```
/subsystem=resource-adapters/resource-adapter=JBossTestCF2:add(archive=genericra.rar,
transaction-support=XATransaction)

/subsystem=resource-adapters/resource-adapter=JBossTestCF2/connection-definitions=
JBossTestCF2:add(jndi-name=java:jboss/JBossTestCF2, enabled=true, allocation-retry=5,
use-java-context=true, security-domain=HsqlDbRealm, blocking-timeout-millis=4000,
class-name= com.sun.genericra.outbound.ManagedJMSCConnectionFactory,
background-validation=false, background-validation-millis=5000,
idle-timeout-minutes=2, allocation-retry-wait-millis=4000)

/subsystem=resource-adapters/resource-adapter=JBossTestCF2/connection-definitions=
JBossTestCF2/config-properties=ConnectionValidationEnabled:add(value=true)

/subsystem=resource-adapters/resource-adapter=XUConnectionFactory:add(
archive=genericra.rar, transaction-support=NoTransaction)

/subsystem=resource-adapters/resource-adapter=XUConnectionFactory/
connection-definitions=XUConnectionFactory:add(enabled=true, use-java-context=true,
jndi-name=java:jboss/XUConnectionFactory, max-pool-size=50, idle-timeout-minutes=15
class-name=com.sun.genericra.outbound.ManagedQueueConnectionFactory,
blocking-timeout-millis=50000)
```

Listing 22: CLI Scripts for Adding the Migrated Resource Adapters Configuration

```
/subsystem=web/virtual-server=jboss.web:add(enable-welcome-root=true,
alias=["company.com", "example.org", "localhost", "example.com"])

/subsystem=web/connector=connector1:add(socket-binding=https, protocol=HTTP/1.1,
scheme=https, secure=true, enabled=true)

/subsystem=web/connector=connector1/ssl=configuration:add(name=ssl, protocol=TLS,
certificate-key-file=${jboss.server.config.dir}/keys/chap8.keystore,
verify-client=false, password=changeit)

/subsystem=web/connector=connector2:add(socket-binding=ajp, protocol=AJP/1.3,
redirect-port=8445, scheme=http, enabled=true)

/subsystem=web/connector=connector3:add(socket-binding=createdSocket, enabled=true,
protocol=HTTP/1.1, redirect-port=8447, scheme=http)

/socket-binding-group=standard-sockets/socket-binding=createdSocket:add(port=8000)
```

Listing 23: CLI Scripts for Adding the Migrated Web Engine Configuration

## Bibliography

- [1] MARCHIONI, Francesco. *JBoss AS5 Development: Develop, Deploy, and Secure Java Applications on this Robust, Open Source Application Server*. Birmingham: PACKT PUBLISHING, 2009. 416 p. ISBN 978-1-84951-678-5.
- [2] MARCHIONI, Francesco. *JBoss AS 7 Configuration, Deployment and Administration*. Birmingham: PACKT PUBLISHING, 2011. 380 p. ISBN 1-849-51678-2.
- [3] JAMES, David – JOHNSON, Peter. *JBoss in Action: Configuring the JBoss Application Server*. Greenwich: Manning Publication Co., 2009. 408 p. ISBN 978-1-933988-02-3.
- [4] GÜLCÜ, Ceki. *Short introduction to log4j*. [online] March 2002. <<http://logging.apache.org/log4j/1.2/manual.html>> [accessed 22 February 2013]
- [5] KHAN, Kabir – LAGROUW, Danny. *Logging Configuration*. [online] 02 February 2011. <<https://docs.jboss.org/author/display/AS71/Logging+Configuration>> [accessed 22 February 2013]
- [6] Wikipedia contributors. *Application server*. Wikipedia, The Free Encyclopedia. [online] 22 February 2013. <[http://en.wikipedia.org/w/index.php?title=Application\\_server&oldid=539638468](http://en.wikipedia.org/w/index.php?title=Application_server&oldid=539638468)> [accessed 27 February 2013]
- [7] LLOYD, David. *Introduction*. [online] 13 July 2011. <<https://docs.jboss.org/author/display/MODULES/Introduction>> [accessed 28 February 2013]
- [8] JBoss Community. *Administration And Configuration Guide*. [online] November 2008. <[http://docs.jboss.org/jbossas/docs/Administration\\_And\\_Configuration\\_Guide/5/html/index.html](http://docs.jboss.org/jbossas/docs/Administration_And_Configuration_Guide/5/html/index.html)> [accessed 1 March 2013]
- [9] BRAUN, Heiko – KHAN, Kabir. *Admin Guide*. [online] 22 July 2011. <<https://docs.jboss.org/author/display/AS7/Admin+Guide#AdminGuide-ConfigurationFiles>> [accessed 4 March 2013]
- [10] Project JAXB. [online] 2013. <<http://jaxb.java.net/2.2.6/docs/ch01.html#documentation>> [accessed 5 March 2013]
- [11] *EclipseLink MOXy*. [online] 2013. <<http://www.eclipse.org/eclipselink/moxy.php>> [accessed 5 March 2013]

- [12] MUCKENHUBER, Emanuel – FINK, Wolf-Dieter. *DataSource configuration*. [online] 11 December 2012. <<https://docs.jboss.org/author/display/AS71/DataSource+configuration>> [accessed 5 March 2013]
- [13] TEXIER, Luc – CLERE, Jean-Frederic. *VersionOfTomcatInJBossAS*. [online] 2 October 2012. <<https://community.jboss.org/wiki/VersionOfTomcatInJBossAS>> [accessed 19 March 2013]
- [14] *JBoss Web Configuration 2.1.x*. [online] <<http://docs.jboss.org/jbossweb/latest/config/index.html>> [accessed 20 March 2013]
- [15] *JBoss Application Server*. [online] <<http://www.jboss.org/jbossas.html>> [accessed 29 March 2013]
- [16] Java Enterprise Edition. [online] <<http://www.oracle.com/technetwork/java/javaee/overview/index.html>> [accessed 29 March]
- [17] Wikipedia contributors. *JBoss*. Wikipedia, The Free Encyclopedia. 22 March 2013. <<http://en.wikipedia.org/w/index.php?title=JBoss&oldid=546382187>> [accessed 29 March 2013]