FIT SCHEDULER – AN ENTERPRISE-LEVEL OPEN SOURCE JOB SCHEDULER

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Venkat Sudheer Reddy Aedama

SPRING
2012
FIT SCHEDULER – AN ENTERPRISE-LEVEL OPEN SOURCE JOB SCHEDULER

A Project

by

Venkat Sudheer Reddy Aedama

Approved by:

__________________________________, Committee Chair
Du Zhang, Ph.D.

__________________________________, Second Reader
Mary Jane Lee, Ph.D.

________________________________________
Date
Student: Venkat Sudheer Reddy Aedama

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Nikrouz Faroughi, Ph.D.

Date

Department of Computer Science
Abstract

FIT SCHEDULER – AN ENTERPRISE-LEVEL OPEN SOURCE JOB SCHEDULER

by

Venkat Sudheer Reddy Aedama

Statement of Problem

Enterprise Job Schedulers and Rule Engines are an expensive class of software. According to an IDC report, the market for job scheduling software licenses in 2009 was 1.5 billion USD. Over the past 25 years, since job schedulers appeared in the market, they compete on price, but not on solutions. Rule Engines are continuously evolving to solve complex problems for many software organizations. Today’s Job Schedulers and Rule Engines typically provide huge set of features with Graphical User Interfaces over a distributed network of computers to monitor the background executions and setup new rules or jobs. Job Schedulers and Rule Engines share a common goal in orchestrating the integration of real-time business activities over different platforms and application environments.
The result of a job execution often needs to be validated against the most probable outcomes. The most probable outcomes can be stored in a Rule Database as understandable rules to support the Rule Engine as a Centralized Knowledge Base, increasing the ease of rules-invocation, consumption, portability and maintenance.

An extensive application would be in an Enterprise-level Job Scheduler performing the ETL (Extract, Transform and Load) process across various database connection pools/multiple applications. As the Job Scheduler only involves in ETL, the data that is loaded needs to be validated against the most probable outcomes. Designing a set of rules to fire after each Load after a job fire validates the data before it is further used in the application. This provides reliable data for the business workflow enhancing the business process as a whole.

Approach

Conclusions Reached

It is a platform independent and vendor independent solution to perform the process of Extraction, Transformation and Data Load within an Application Server across multiple applications synchronously. It leverages a common framework to manage the connection pools to achieve data synchronization. Aimed at providing a configuration-driven approach to the end-user, the functionality of FIT Scheduler is highly extendable in future. It has built in features to overcome Daylight Saving Time, evaluate the data loaded using a custom-built rule engine, re-run failed jobs, launch new jobs, monitor the scheduling system, increase scheduler resources on the fly etc. It acquires various powerful features from Quartz like fail-over, load balancing, and clustering.

______________________, Committee Chair
Du Zhang, Ph.D.

______________________
Date
PREFACE

“Necessity is the mother of invention”

-Plato

This report is my masters’ project for the culmination of my Master program in Computer Science, California State University, Sacramento. Moreover, it is also a culmination of my internship at California Independent System Operator (California ISO), Folsom. I really appreciate many people who helped me to learn various booming technologies in the ever-changing technological world.

I would like to firstly thank my mentor, Steve Rylander at California ISO for giving me the opportunity to learn, taking the time to show me the necessary skills and accepting me as I am with all my eagerness and my sheer joy over the little things I achieved every day, never squelching that spirit in me.

I also want to extend my gratitude to Prof. Du Zhang for encouraging me to incorporate the rule engine to make FIT Scheduler a FIT one! I want to thank you for having enough faith to share your work, ideas and vision.

I will continue to challenge myself in the future. This is not the end. It is only the beginning.
DEDICATION

Dedicated to my mentor
Steve Rylander
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>viii</td>
</tr>
<tr>
<td>Dedication</td>
<td></td>
<td>ix</td>
</tr>
<tr>
<td>List of Tables</td>
<td></td>
<td>xii</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td>xiii</td>
</tr>
<tr>
<td></td>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Abstract</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2. BACKGROUND STUDY</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Preliminary Investigations</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2.3 Review of Research</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>3. SYSTEM ARCHITECTURE</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>3.1 Architecture of Web-service layer</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>3.2 Architecture of Web-based Graphical User Interface</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>3.3 System Requirements</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>4. DESIGN AND IMPLEMENTATION</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>4.1 Database Modeling and Design</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>4.2 Design of Web-services</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>4.3 Design of Rule Engine</td>
<td></td>
<td>41</td>
</tr>
</tbody>
</table>
4.4 User Interface Design .......................................................................................... 44

4.5 System Implementation ....................................................................................... 46

5. RESULTS AND PERFORMANCE .................................................................. 48

5.1 Comparison Tests .............................................................................................. 48

5.2 Performance Analysis ......................................................................................... 49

6. CONCLUSIONS AND FUTURE SCOPE .................................................... 53

Appendix A. JBoss Configuration with Quartz ......................................................... 55

Appendix B. Quartz Service Code Modules ........................................................... 64

References .............................................................................................................. 77
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sample Web-services</td>
<td>40</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Quartz Service Architecture</td>
<td>19</td>
</tr>
<tr>
<td>2. User Interface Interaction with Application Server</td>
<td>22</td>
</tr>
<tr>
<td>3. Class Diagram for a Probe</td>
<td>34</td>
</tr>
<tr>
<td>4. Flow-chart diagram to create a new probe</td>
<td>38</td>
</tr>
<tr>
<td>5. Design of Top Panel</td>
<td>45</td>
</tr>
<tr>
<td>6. Design of Bottom Panel</td>
<td>45</td>
</tr>
<tr>
<td>7. System Implementation</td>
<td>47</td>
</tr>
<tr>
<td>8. Data Load ID vs. Elapsed Time for one Measurement Queried for each load</td>
<td>50</td>
</tr>
<tr>
<td>9. Data Load ID vs. Elapsed Time for four Measurement Queried for each load</td>
<td>51</td>
</tr>
<tr>
<td>10. Measurements Queried vs. Elapsed Time in milliseconds</td>
<td>51</td>
</tr>
<tr>
<td>11. JBoss Standard Logging for Quartz Properties Deployment</td>
<td>61</td>
</tr>
<tr>
<td>12. Project setup for Quartz Service</td>
<td>69</td>
</tr>
</tbody>
</table>
CHAPTER 1: INTRODUCTION

1.1 Abstract:

Statement of Problem

Enterprise Job Schedulers and Rule Engines are an expensive class of software. According to an IDC report, the market for job scheduling software licenses in 2009 was 1.5 billion USD. Over the past 25 years, since job schedulers appeared in the market, they compete on price, but not on solutions. Rule Engines are continuously evolving to solve complex problems for many software organizations. Today’s Job Schedulers and Rule Engines typically provide huge set of features with Graphical User Interfaces over a distributed network of computers to monitor the background executions and setup new rules or jobs. Job Schedulers and Rule Engines share a common goal in orchestrating the integration of real-time business activities over different platforms and application environments.

The result of a job execution often needs to be validated against the most probable outcomes. The most probable outcomes can be stored in a Rule Database as understandable rules to support the Rule Engine as a Centralized Knowledge Base, increasing the ease of rules-invocation, consumption, portability and maintenance.
An extensive application would be in an Enterprise-level Job Scheduler performing the ETL (Extract, Transform and Load) process across various database connection pools/multiple applications. As the Job Scheduler only involves in ETL, the data that is loaded needs to be validated against the most probable outcomes. Designing a set of rules to fire after each Load after a job fire validates the data before it is further used in the application. This provides reliable data for the business workflow enhancing the business process as a whole.

Approach
FIT stands for Fully Integrated Tracking. FIT Scheduler is a lightweight open-source solution for organizations requiring client-server Enterprise Job Schedulers who cannot bear the over-priced licenses. FIT Scheduler uses Quartz Scheduling Framework as back-end, Google Web Toolkit framework combined with Sencha’s Ext-GWT library for the web-based front-end and various other open-source projects like Jersey implementation of Java API for RESTful Web-services, Groovy programming language, Apache POI – API for Microsoft Documents.

Conclusions Reached
It is a platform independent and vendor independent solution to perform the process of Extraction, Transformation and Data Load within an Application Server across multiple applications synchronously. It leverages a common framework to manage the connection pools to achieve data synchronization. Aimed at providing a configuration-driven
approach to the end-user, the functionality of FIT Scheduler is highly extendable in future. It has built in features to overcome Daylight Saving Time, evaluate the data loaded using a custom-built rule engine, re-run failed jobs, launch new jobs, monitor the scheduling system, increase scheduler resources on the fly etc. It acquires various powerful features from Quartz like fail-over, load balancing, and clustering.

Rest of the report is organized as follows:
Chapter 2 discusses the background of the project, a brief introduction to Quartz Scheduling Framework, and preliminary investigations on enterprise-level job schedulers, related projects and their limitations ending with a review of the research of the related work.

Chapter 3 discusses the System Architecture of FIT Scheduler including the architecture of web-service layer and user interface interaction with the web-service layer along with a brief introduction of RESTful Web-services and Groovy. The software specifications are given at the end of this chapter.

Chapter 4 discusses the System Design and Implementation. This chapter includes database modeling, web-services design and user interface design. The database modeling includes design of database models for jobs, their configuration, rule engine and object-oriented mapping model. The web-services design includes designing web-
service layer on top of Quartz Scheduling Framework. The user interface design includes designing the FIT Scheduler’s Graphical User Interface in a single web page!

Chapter 5 discusses the performance analysis and provides comparisons between FIT Scheduler and other enterprise-level job schedulers.

Chapter 6 concludes the report giving information on future scope of FIT Scheduler and rule engine.

The Appendix A contains information on configuring FIT Scheduler with JBoss Application Server and Appendix B provides the web-service layer code modules.
CHAPTER 2: BACKGROUND STUDY

A basic feature of job scheduler is to define and monitor the execution of workflows by prioritizing the queue to control the order of submission or execution of jobs. Various modern Operating Systems and batch systems have been providing reliable job scheduling platforms over decades in the form of Batch-processing, Event-driven automation and Service Oriented job scheduling.

There are various proprietary solutions and open-source solutions for enterprise-level job scheduling. In large organizations, requirement of Job Schedulers has become very intense for various purposes like computation, simulation, data analysis, visualization, event processing, IT housekeeping etc. Organizations choose their Job Scheduler product considering a wide variety of features depending upon their requirement and budget.

The requirement for client-server job schedulers is relatively high than peer-to-peer systems. In the Enterprise Java World, some of the most popular and open source job schedulers/frameworks are Quartz, GNUbatch, cron4j, Essiembre J2EE Scheduler, The Maui Job Scheduler etc. There are some proprietary players like IBM Tivoli Workload Scheduler, Tidal Enterprise Scheduler, and BMC Control-M. Some of them are out-dated and over-priced, yet the market share is surprisingly high!
California ISO is a utilities organization responsible for power transmission throughout the State of California. One of the primary duties of California ISO is to forecast power requirement throughout the State using its forecasting tools at regular intervals. The forecasting tools provide forecasted data in various formats like Spreadsheets or database tables. This data is loaded to the Graphical User Interfaces of generators so that they ramp the generation capacity to satisfy the power requirement.

California ISO currently uses “cron” [12] utility that comes with the Unix Systems to perform the data loading process of the forecasted data from the generators. They have several Perl scripts for each forecasting tool. The major problem is a change in the forecasting tool requires deep knowledge in Perl, as they need to change several Perl scripts. The requirement of a service layer on top of the scheduling framework to perform the ETL process (Extract, Transform and Load) of the forecasted data irrespective of the future changes in the forecasting tool, leveraging a common framework across various connection pools, providing a configuration-driven approach to the user gave the roots for the need of FIT Scheduler.

Sometimes the forecasting tools deliver bad data. As the data lacks validation process, it is directly supplied to the generators. This results in a huge budget loss as California ISO spends more money to satisfy the requirement to overcome blackouts. This gave the roots for a rule engine to validate the data before it is supplied to the generators.
Introduction to Quartz:

Quartz is a powerful open-source job-scheduling framework licensed under Apache 2.0 license [1]. Quartz is one of the most widely used frameworks for building enterprise job schedulers. Vodafone, U.S. Department of Defense, Atlassian, Cisco, Apache Jakarta, Spring, JBoss [2], and Adobe are a few of the thousands of Quartz users. Quartz runs under various run-time environments or embedded within another standalone application or instantiated from an application server or as a cluster of standalone programs or individual standalone program (via Remote Method Invocation).

2.1 Preliminary Investigations:

1. Areas of application:


2. Scheduling Frameworks:

   There are various open-source job-scheduling frameworks to support web-based solutions in J2EE environment like Quartz, cron-4j, jBatchEngine, MyBatchFramework. The most powerful and widely used open-source framework in J2EE environment is Quartz. FIT Scheduler uses Quartz Scheduling Framework for building the Quartz Service, a service layer on top of Quartz API.
3. **Web-services and styles of use:**

Web-services support machine-machine interaction over the network. There are three most common styles of using web-services. They are Remote Procedure Call (RPC), Service Oriented Architecture (SOA) and REpresentational State Transfer (REST). We use RPC and REST, as they are lightweight and easy to develop.

4. **Supportive platforms and framework dependencies:**

Designing an OS-independent solution is an important criterion. Some job schedulers run on only on Windows platforms while some run only on UNIX. Some schedulers use UNIX based cron while some others use third party scheduling frameworks. Very few of them are cross-platform (.NET platforms, J2EE and stand-alone). As we design a web-based job scheduler, an Application Server is the main platform for the scheduler.

There are various Application Servers available in the market like JBoss [2], Apache Tomcat, Oracle’s WebLogic, and IBM’s WebSphere etc. JBoss is an open-source Java Application Server. It is exceptionally lightweight, strictly compliant, and easily testable delivering solutions with unparallel speed. JBoss easily integrates with various scheduling frameworks. JBoss also ships with Quartz Scheduling Framework. However, we install the latest stable version of
Quartz inside JBoss, as the Quartz that ships with JBoss is a very old version. We learn more about installing Quartz in JBoss in further chapters.

5. **Common Graphical User Interfaces and Front-end Frameworks:**

The most common Graphical User Interfaces are either Desktop-based or Web-based. However, the demand for Web-based job schedulers is comparatively high than Desktop-based solutions.

There are various front-end frameworks for Java platform. Some of the notable ones are Google Web Toolkit, Java Server Faces, Spring, Grails, Apache Tapestry, and Apache Wicket etc. Google Web Toolkit is a powerful front-end framework vastly used to design AJAX enabled web applications. Google Web Toolkit leverages the powerful widgets of Sencha’s Ext-GWT library very easily. The soul of GWT is its Java to JavaScript compiler. Developers can build the User Interface of a web application programming Java. Google Web Toolkit compiles the Java code to HTML and .js files. However, the server-side code need not be Java.

6. **Programming languages:**

Designing the job scheduler in a J2EE environment involves development in languages that support Java Virtual Machine (JVM). Apart from Java language
itself, there are various well-known JVM languages like Groovy, Scala, JRuby, Jython, AspectJ etc. We use Java and Groovy for development of the web-based front-end and Groovy for building the Quartz Service. Groovy provides many modern programming features like closures, powerful processing primitives with almost-zero learning curve for Java Developers.

7. **Performance:**

High availability, load-balancing, fail-over capabilities are very important for an Enterprise-level Job Scheduler. Quartz Scheduling Framework supports all these features with additional features like clustering, job persistence in database etc.

2.2 Related Work:

1. **JWatch – A Quartz Monitor**

JWatch [8] is a web-based, open-source, real-time monitoring solution for Quartz Scheduling Framework. JWatch provides various features like monitoring multiple instances of Quartz across various applications concurrently. It provides useful information about Quartz Jobs and Triggers associated with them. JWatch comes with RESTful JavaScript Object Notation (JSON) API for developers who wish to extend the application. The current version of JWatch is 0.8. It supports Quartz 2.X versions only. JWatch is planning to release functionality with older
versions of Quartz. JWatch code license is GNU Lesser GPL (General Public License).

**Limitations of JWatch:**

a. The current version - JWatch 0.8 is not completely ready for clustered environments.

b. JWatch is capable of running only in Apache Tomcat Servlet Container. It does not support Glassfish or JBoss Application Servers.

c. JWatch embeds the Quartz Properties file inside the deployable Web-application archive (war file). This is not only a security threat but also lacks the flexibility to change Quartz resources on the fly. For example, if the user wants to schedule a thousand jobs and the number of threads specified in the Quartz Properties file is five then many jobs go in the waiting state. To change the properties, the user has to rebuild JWatch to support the thousand jobs!

d. It has dependencies on Java Management Extensions (JMX) libraries and several other libraries like EZMorph, XStream etc. making it a heavy deployment.

e. The front-end and back-end are tightly coupled which makes the future extensions difficult to implement.

f. The front-end uses Java Servlet Pages (JSP), JavaScript, Cascading Style Sheets and HTML. The learning curve for future developers is high, as they need to learn these languages.
2. **Citrine-scheduler**

Licensed under Apache 2.0 License [1]. Citrine [9] is a Java based web-application to configure, manage and monitor various tasks. The tasks are usually shell scripts. It also supports Groovy scripts as tasks. Citrine also uses Quartz Scheduling Framework for running the tasks. It ships with a web-based graphical user interface. It is more like a Graphical User Interface replacement for UNIX Cron with additional functionality. It is one of most matured solutions using Quartz Scheduling Framework. Last.fm [7], a music website uses Citrine Scheduler for commercial purposes. The current version of Citrine is 4.1.0.

**Limitations of Citrine Scheduler:**

a. It is database platform dependent and currently it supports only MySQL database platform.

b. The logging mechanism is very basic. Citrine cannot handle jobs generating hundreds of log files very well. It requires extensive modifications to make the log files to store in separate directories for each task.

c. Citrine Scheduler is a very heavyweight deployable file. It has dependencies on numerous libraries like spring, hibernate, Java Transaction API (JTA), Java Persistence API (JPA), Apache ivy, FreeMarker, Mockito.

d. The front-end and back-end for Citrine Scheduler are also tightly coupled making it difficult for others to extend or use the project by adding custom changes.
e. Citrine also uses Java Servlet Pages (JSP), JavaScript, Cascading Style Sheets and HTML making the learning curve of future developers not so easy.

3. MySchedule

Licensed under Apache 2.0 license [1], MySchedule [10] is a web-application for managing multiple Quartz Schedulers. It runs on any Java Servlet Container. It has a friendly user interface to create, modify and delete multiple instances of Quartz Scheduler configurations. The latest version of MySchedule is 2.4.2.

**Limitations of MySchedule:**

a. MySchedule also does not support changing Quartz Properties on the fly.

b. The users have to load the job scheduling data in XML format and upload it to server to feed the job configuration for the scheduler. This requires the end user to have knowledge in writing XML.

c. MySchedule does not ship with any jobs. The users code the tasks in Java and supply to the scheduler.

d. MySchedule faces the same limitations of tight coupling of front-end and back-end. It also uses Java Server Pages (JSP), JavaScript, CSS and HTML for the front-end design.

2.3 Review of Research

This topic covers a systematic review of the limitations of the existing products and solutions to overcome these limitations.
1. **Requirement to change Quartz Scheduler Properties on the fly:**

The properties file contains various properties defining a Quartz Scheduler instance. Some of the properties are scheduler instance name, scheduler instance ID, thread count, thread priority, job store mechanism, Data Source name for Quartz, JNDI (Java Naming Directory Interface) URL (Uniform Resource Locator) etc. The users of the job scheduler may need to change any of these properties or create new properties.

2. **Security:**

The properties file of Quartz Scheduler should not be bundled with the deployment file as it raises various security issues like exposing the Quartz Data Source name, JNDI URL, scheduler name, instance id etc to the developer. It is secure to keep the Quartz Properties file under the control of Application Server Administrator.

3. **Lose-coupling between the web interface and back-end scheduling system:**

The sound principle of web-application development is lose-coupling between the user-interface and back-end. This principle helps the future developers to replace either of the ends with new frameworks while using at least one of them. If the users want to replace the Quartz Scheduling Framework with a new scheduling framework, they can replace it while using the same user-interface. If a non-
GWT, developer wants to extend FIT Scheduler, s/he can replace the front-end framework with a new framework by leveraging the Quartz Service.

4. **Database platform independency:**

To provide independency on the database platform there are various solutions. The best way is to write plain SQL on the server side. We use Oracle Database Server as database platform. Since we use plain SQL, almost all the widely used database servers like MySQL, Oracle and Apache Derby have similar syntax for basic operations like CREATE, SELECT, DELETE and UPDATE.

5. **Application server independency:**

Developing an Application Server-independent Enterprise-level Job Scheduler is practically not possible. There is a wide range of Application server platforms. JBoss Application Server and Apache Tomcat Servlet Container are most widely used Servlet containers/application servers. The development of FIT Scheduler aims at deployment inside JBoss Application Server as well as Tomcat Servlet Container.

6. **Lightweight solution:**

Organizations focus a lot on developing lightweight software in an agile environment because of their increased effectiveness over the past years. Trying
to avoid extra libraries by building custom solutions helps the developer to overcome adding another black box in the deployable file, thus giving a lightweight output.

7. **Steep learning curve for the developers who wish to extend the application:**

   It is always desirable to write an application in a single programming language rather than multiple languages as it makes the learning curve difficult for the developers who want to extend the application. The existing solutions use Java, HTML, JavaScript, CSS, SQL and other persistence frameworks like the Java Persistence API, Hibernate etc. The web-based user interface and the service layer on Quartz use Java and Groovy only. A Java developer has almost-zero learning curve for learning Groovy. Surprisingly, a .java file when renamed with .groovy extension, the program acquires Groovy properties!
FIT Scheduler comprises of two components namely Quartz Service and Web-based Graphical User Interface. We separate the back-end and front-end to provide lose-coupling between the front-end and back-end. We discuss the architecture of these two components separately and then combine both these components and explain how they work together.

One of the aims of building web-services on Quartz is to separate the web-based graphical user interface from the scheduling system. This provides the future developers a great facility to replace back-end Quartz framework with some other scheduling framework or the front-end GWT framework with other ones like Ruby or Grails.

**Introduction to RESTful web-services:**

We use RESTful web-services as the platform of web-services as it is lightweight than SOAP based standards. In addition, RESTful web-services do not require XML parsing and consumes less bandwidth unlike SOAP, which requires a header for every message. RESTful web-services are stateless. We use mainly GET operations provided by REST architecture. Hence, all the operations are safe!

We use Jersey implementation of JAX-RS [5] (JSR 311) (Java API for RESTful Web-services) as it is open-source and one of the best production quality to implement
RESTful web-services. We can download the latest version of Jersey from their official web-site.

Introduction to Groovy:
The programming language we use for building Quartz Service is Groovy. Groovy is a powerful dynamic language for the Java platform. It supports various modern programming features with an almost-zero learning curve. It simplifies testing, improves production, and integrates with all Java classes and libraries. It is more like a super-version of Java.

3.1 Architecture of Web-service layer:
Quartz Service is the Web-service layer involving development of Web-services on top of Quartz Scheduling Framework. We use RESTful (REpresentational State Transfer) Web-services to build the service layer. REST is an architectural style for distributed hyper-media systems. A Web-service confirming to the architectural principles of REST is a RESTful Web-service.

REST style architecture involves servers and clients. Clients send requests to server. The server processes these requests and return appropriate responses. Client Requests and Server Responses are the basis of transfer of representations of resources. A resource is an entity, item or the thing that we want to expose.
In Quartz Service, the resource that we want to expose is the Quartz Application Package Interface (API). Quartz Service is all about interacting with Quartz API, specifically Quartz Jobs through a Web-service. The interaction involves creation of new jobs, deleting, scheduling, unscheduling, pausing, resuming, re-submitting failed jobs, clearing all jobs from scheduler etc. Quartz API interacts with the Quartz database using a data source. We create a data source separate for Quartz API interaction with the Oracle database.

Figure 1: Quartz Service Architecture
Components in Quartz Service:

1. **Quartz Database Tables:** Persistence of the job information in a database is very essential as we use this information to feed the web-based Graphical User Interface of the FIT Scheduler. Quartz achieves persistence using JDBCJobStore mechanism inbuilt in Quartz API.

2. **Quartz Properties File:** We configure the Quartz Scheduling Framework using a properties file. This properties file contains all the configuration details. We can embed the Quartz Properties File inside the Quartz Service application or separately deploy in the Application Server. We deploy Quartz Properties File as a managed bean inside JBoss Application Server. We cover more information in the further topics.

3. **Quartz DataSource:** Basically, a DataSource is a facility for storing data. Quartz API accesses the Quartz Database Tables using the Quartz DataSource to perform database operations. Quartz official documentation suggests never to up-date Quartz Tables directly from the database. We use Quartz API to update the information in Quartz Database. We register this DataSource object in the server and provide access to it by providing the related information in the Quartz Properties File.
4. **RESTful Web-services:** This component involves wrapping Quartz API in Web-services. Any client, even a web-browser with Ajax and JavaScript can consume these Web-services. We discuss building Web-services in further chapters.

3.2 **Architecture of Web-based Graphical User Interface:**

We use Remote Procedure Call (RPC) Web-services for communication between the Graphical User Interface and RESTful Web-services. Indirectly, we use RPC to call RESTful Web-services. This feature allows lose-coupling between the front-end and the back-end.
Components in Web-based Graphical User Interface:

1. **Google Web Toolkit (GWT) components**: We use GWT framework to build the front-end. GWT has a rich library for components like panels, windows, containers, canvases, buttons, grids, and various kinds of layouts for these widgets. GWT comes with an RPC (Remote Procedure Call) framework based
on Java Servlet Architecture. This makes it easy for client and server components to exchange Java objects over HTTP.

2. **Ext-GWT Sencha Library components:** We leverage GWT with the powerful Sencha’s JavaScript library - Ext-GWT. Ext-GWT takes GWT to the next level with their feature rich templates, layouts, high performance widgets, cross-browser compatibility etc.

3. **Custom components:** GWT and Ext-GWT provide various widgets and layouts to build the Web-based Graphical User Interface, but we build some custom components like Wizard Windows, Toolbars etc. using the primitive components of GWT and Ext-GWT like Content Panel, Layout Container, Window, Button Bar, Card Panel, and various layouts like Fit Layout, Card Layout, and Flow Layout etc.

3.3 **System Requirements:**

This section gives list of recommended prerequisites required for FIT Scheduler in two sub-sections titled Hardware Requirements and Software Requirements.

3.3.2 **Hardware Requirements:**

**Architecture:** 32-bit or 64 bit Operating System
Processing Power: 2 GHz Clock Speed, 2 MB CPU Cache,

Memory: Minimum 4 GB SD/ DDR RAM

Secondary Storage: 20 GB Hard Disk

Peripherals: Keyboard, Mouse

3.3.2 Software Requirements:

Java Platform: Java Virtual Machine, Java Development Kit 6

Database Platform: Oracle 11g R2

Application Server: JBoss 5.1.0 GA

Web-browser: Any Ajax enabled web-browser with JavaScript support.

Development Environment: Eclipse IDE for Java EE Developers (Indigo)

GUI Framework: Google Web Toolkit 2.4.0, Sencha Ext-GWT 2.2.5

Scheduling Framework: Quartz 2.0.2

Web-service Library: Jersey JAX-RS 1.12 (Java API for RESTful Web-services)

Programming languages: Java 6, Groovy 1.8.6, SQL

Build Tool: Apache Ant Builder

Other third party libraries: Apache POI 3.8, Apache Commons IO 2.1, gwt-log 3.1.7, log4j 1.2.14
4.1 Database Modeling and Design:

One of the main aims of FIT Scheduler is to provide a configuration driven functionality to the end-user. Each job has a specific configuration. The job along with its configuration is termed as a “Probe”. FIT Scheduler supports data loading from database sources as well as spreadsheets. The jobs that perform data loading from database as sources are termed as “Database Probes” while the jobs that use spreadsheets as sources are termed as “Spreadsheet Probes”. Both kinds of probes have different configurations. However, there are a few common configuration details as well.

4.1.1 Design of database model for monitoring jobs:

Monitoring jobs involves provision to view the audit information related to each data loading job run. We design an audit table as follows:

Create table FIT_DATA_LOADS_AUDIT

(DATA_LOAD_ID INTEGER,
DELETED_DATA_LOAD_ID INTEGER,
REPLACEMENT_DATA_LOAD_ID INTEGER,
DATA_PROBE_NAME VARCHAR2(255),
GIVEN_START_DATE VARCHAR2(255),
GIVEN_HOUR_STARTING INTEGER,
GIVEN_END_DATE VARCHAR2(255),)
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIVEN_HOUR_ENDING</td>
<td>INTEGER</td>
</tr>
<tr>
<td>DATA_PROBE_ID</td>
<td>INTEGER</td>
</tr>
<tr>
<td>DATA_PROBE_TYPE</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>DATA_PROBE_DESCRIPTION</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>TIMESTAMP_COLUMN_NAME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>DATA_INTERVAL</td>
<td>INTEGER</td>
</tr>
<tr>
<td>OFFSET</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SOURCE_NAME</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>SOURCE_QUERY</td>
<td>VARCHAR2 (4000)</td>
</tr>
<tr>
<td>SOURCE_JNDI_NAME</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>CONFIG_JNDI_NAME</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>TARGET_JNDI_NAME</td>
<td>VARCHAR2 (255)</td>
</tr>
<tr>
<td>RUN_START_TIME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>RUN_END_TIME</td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>ELAPSED_RUN_TIME</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ROWSEXPECTED</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ROWS_QUERIED</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MEASUREMENTS_QUERIED</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ROWS_INSERTED</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ROWS_DELETED</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ESTIMATE_FLAG</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>
START_TIME VARCHAR2 (30),
END_TIME VARCHAR2 (30),
SUCCESS_FLAG VARCHAR2 (8),
 CONSTRAINT FIT_DATA_LOADS_AUDIT_PK PRIMARY KEY
 (DATA_LOAD_ID)
);

Below is the explanation for each attribute:

1. Data Load ID: The Data Load ID for a probe is loaded from an Oracle Sequence. It is a unique value for each job run.

2. Deleted Data Load ID: The Deleted Data Load ID for a probe is the Data Load ID of another data load rerun.

3. Replacement Data Load ID: The Replacement Data Load ID for a probe is the Data Load ID of the probe that reran the probe.

4. Data Probe Name: It is same as the job name.

5. Given Start Date: It is the start date schedule configured for the probe. This contains the day, month and year fields.

6. Given Start Hour: It is the start hour schedule configured for the probe. This contains hour and minute fields.

7. Given End Date: It is the end date schedule configured for the probe. This contains day, month and year fields.
8. Given End Hour: It is the end hour schedule configured for the probe. This contains hour and minute fields.

9. Data Probe ID: The Data Probe ID for a probe is loaded from an Oracle Sequence. It is also a unique value for each Probe

10. Data Probe Type: This is the job type.

11. Data Probe Description: This is the job description.

12. Timestamp Column Name: This is the name of the timestamp column in the data source, spreadsheet or database table.

13. Data Interval: This is the time interval between two successive intervals of source data.

14. Offset: This is the time offset by the user with the fire-time of the job to start the data load.

15. Source Name: This attribute holds the source spreadsheet name. For a database probe, this attribute value is null.

16. Source Query: This is the SQL query the user wants to run for each data load. For a spreadsheet probe, the value of this attribute is null.

17. FIT JNDI Names: We create three JNDI names for FIT Scheduler: Source JNDI, Config JNDI and Target JNDI. We use them as follows:
   - Source JNDI name to perform look-up on the source database tables
   - Config JNDI name to perform look-up on the configuration database tables
   - Target JNDI name to perform look-up on the destination database table
18. Data Load Start Time: This is the timestamp value when the probe starts loading the data from a source database or spreadsheet. The precision of this timestamp is up to milliseconds.

19. Data Load End Time: This is the timestamp value when the probe completes loading the data from a source database or spreadsheet. The precision of this timestamp is also up to milliseconds.

20. Elapsed Time: The difference between the above two timestamp values is the result of elapsed time.

21. Rows Expected: The user may expect a particular number of rows for each data load.

22. Rows Queried: The number of rows queried by the scheduler for a job run.

23. Measurements Queried: This is the number of columns that Scheduler queries for each data load from the source database or spreadsheet.

24. Rows Inserted: This attribute stores the number of rows inserted for each data load.

25. Rows Deleted: This attribute stored the number of rows deleted if a data load is reran.

26. Estimate Flag: If the user estimates a number of rows, and if the user is sure about the number of rows then this flag can be set. If the user is sure about the number of rows to be loaded for each data load then the value of this flag is 1, otherwise 0.
27. Start Time: The data load interval’s start time.

28. End Time: The data load interval’s end time.

29. Success Flag: If the rows expected is equal to the rows inserted and if the estimate flag is set, then the success flag is equal to one. If the rows expected is not equal to rows expected, and if the estimate flag is not set then the success flag is one. The success flag is zero if and only if the rows expected is not equal to the rows inserted while the estimate flag is set or if the rows loaded for a data load is null.

We need to store the notes of each load and any exceptions that occur during each data load. We create two separate tables to store this information. They are as follows:

Create table FIT_NOTES

(DATA_LOAD_ID INTEGER,
NOTES VARCHAR2 (4000),
CONSTRAINT FIT_NOTES_PK PRIMARY KEY (DATA_LOAD_ID))

Create table FIT_DATA_LOAD_EXCEPTIONS

(DATA_LOAD_ID INTEGER,
EXCEPTION_TEXT VARCHAR2 (4000),
EXCEPTION_TEXT VARCHAR2 (4000),
CONSTRAINT FIT_DATA_LOAD_EXCEPTIONS_PK PRIMARY KEY (DATA_LOAD_ID))
CONSTRAINT FIT_DATA_LOAD_EXCEPTIONS_PK PRIMARY KEY
(DATA_LOAD_ID)
);

4.1.2 Database Modeling for storing Probe Configuration:
This topic discusses the database model of the configuration details specified by the user for each probe. We create three database tables to hold the configuration of a probe. The first table is PROBE_CONFIG to hold all the generic configuration details of each probe. The generic configuration includes attributes for job details like job name, job type, job description and other configuration parameters like rows expected for each data load, offset from job fire time, estimate flag, data interval, time zone of source data etc. The attributes of this table are generic for both the probe types.

We create a database table to hold the generic configuration as shown below:

Create table PROBE_CONFIG

(PROBE_NAME VARCHAR2 (255),
PROBE_TYPE VARCHAR2 (30),
PROBE_ID INTEGER,
PROBE_DESCRIPTION VARCHAR2 (4000),
ROWS_EXPECTED INTEGER,
OFFSET_FROM_FIRETIME INTEGER,
ESTIMATE_FLAG INTEGER,
DATA_INTERVAL INTEGER,
SOURCE_TIMEZONE VARCHAR2 (20),

CONSTRAINT PROBE_CONFIG_PK PRIMARY KEY (PROBE_ID),
CONSTRAINT PROBE_NAME_UNIQUE UNIQUE (PROBE_NAME)
);

The other two tables namely, DATABASE_PROBE_CONFIG and SPREADSHEET_PROBE_CONFIG hold the specific configuration details of the probe types. As the name says, DATABASE_PROBE_CONFIG table holds all the details specific to the database probes while SPREADSHEET_PROBE_CONFIG table holds all the details specific to the spreadsheet probes. The entity relationship diagrams for both the tables are in figures respectively:

Create table SPREADSHEET_PROBE_CONFIG

(SPREADSHEET_PROBE_ID INTEGER,
SPREADSHEET_PROBE_NAME VARCHAR2 (50),
SPREADSHEET_NAME VARCHAR2 (255),
SHEET_NUMBER INTEGER,
SOURCE_TIMESTAMP_COLUMN_NAME VARCHAR2 (255),
SOURCE_TIMESTAMP_COLUMN_FORMAT VARCHAR2 (255),
START_ROW INTEGER,
END_ROW INTEGER,
CONSTRAINT XL_PROBE_CONFIG_PK PRIMARY KEY (SPREADSHEET_PROBE_ID),
CONSTRAINT XL_PROBE_CONFIG_FK FOREIGN KEY (SPREADSHEET_PROBE_ID) REFERENCES PROBE_CONFIG (PROBE_ID),
CONSTRAINT XL_PROBE_NAME_UNIQUE UNIQUE (SPREADSHEET_PROBE_NAME)
);

Create table DATABASE_PROBE_CONFIG

(DATABASE_PROBE_ID INTEGER,
DATABASE_PROBE_NAME VARCHAR2 (50),
SOURCE_JNDI_NAME VARCHAR2 (30),
SOURCE_TIMESTAMP_COLUMN_NAME VARCHAR2 (255),
SOURCE_QUERY VARCHAR2 (4000),
CONSTRAINT DB_PROBE_CONFIG_PK PRIMARY KEY (DATABASE_PROBE_ID),
CONSTRAINT DB_PROBE_CONFIG_FK FOREIGN KEY (DATABASE_PROBE_ID) REFERENCES PROBE_CONFIG (PROBE_ID),
CONSTRAINT DB_PROBE_NAME_UNIQUE UNIQUE (DATABASE_PROBE_NAME)
);


Figure 3: Class Diagram for a Probe
4.1.3 Design of custom Object Relational Mapping Model:

The custom Object Relational Mapping aims at providing independence on naming the columns of the source data. In many applications, there is a requirement of changing the column names in database tables or spreadsheets. Besides this, another common requirement is various data sources having same attribute with a different name. To overcome these two problems, we design an Object Relational Mapping model.

In this model, we map the source columns with unique attribute ids. For example, if a data source has a timestamp column with name TIME_INTERVAL and another data source has a timestamp column with name TIMESTAMP and both mean the same. In this case, we map both these columns to a same id. In future, if the name of this timestamp column changes in either of the database, the end user need not bother about changing the name of the source columns in FIT Scheduler.

We design a database table to hold this mapping configuration as below:

```
Create table FIT_COLUMN_CONFIG

(PROBE_NAME_MAPPING_FOR_COLS VARCHAR2 (255),
SOURCE_COLUMN VARCHAR2 (255),
TARGET_ATTRIBUTE_ID NUMBER,
CONSTRAINT FIT_COLUMN_CONFIG_PK PRIMARY KEY
(PROBE_NAME_MAPPING_FOR_COLS, TARGET_ATTRIBUTE_ID)
);
```
We also design a destination table to stores all the loaded data as shown below:

```sql
CREATE TABLE FIT_DATA
(
DATA_LOAD_ID            INTEGER         NOT NULL,
SOURCE_TIMESTAMP       VARCHAR2 (255),
ATTRIBUTE_ID           INTEGER,
ATTRIBUTE_VALUE        VARCHAR2 (255)
);
```

The SOURCE_TIMESTAMP value contains the timestamp value for which the data load takes place. The ATTRIBUTE_ID corresponds to the id of the column that is loaded. The attribute value is the actual value of the row or cell.

### 4.2 Design of Web-services:

In this topic, we discuss design of Web-services on top of Quartz Scheduling Framework. We use RESTful Web-services to create the service-layer on top of Quartz API. In a RESTful web-service the verbs PUT, POST, GET and DELETE constitute the functionalities of the architecture protocol. GET operations are the safest operations in a Web-service as we do not POST, PUT or DELETE any server resources. We use only GET operations to provide all the functionalities to the end user keeping safety as the first priority as it does not produce any side effects.

The end-user needs various functionalities for the scheduling system like creating new jobs, deleting unwanted jobs, pausing, resuming, re-running failed jobs, querying the
information of a job, clearing the scheduler etc. We design Web-services to support all these needs.

We design RESTful web-services easy to guess.

**List of operations built as Web-services on top of Quartz:**

1. **Create new job**

   We create a new job by passing the following combinations of information in the web-service:

   a. Job name

   b. Job name, start date, start time and repeat interval

   c. Job name, start date, start time, repeat interval, end date and end time

   A job created with no end date and no end time, repeats forever until the job is paused or deleted or the scheduler is cleared. The start time should be at least a minute in future and end time at least five minutes in future. The job name should be unique without spaces or special characters other than (! @, $, ^, &, *, _, +, -, =, ), (, , ]).

   We show a flow chart diagram to create a new probe in the figure:
Figure 4 Flow-chart diagram to create a new probe.

1. **Start**

   - Click Add button on FIT Probes toolbar

2. **Step 1: Enter Job Configuration Details**

3. **Job Type?**
   - Database Probe
   - Spreadsheet Probe

4. **Step 2: Enter Database specific configuration details**

5. **Setup Rules?**
   - **Yes**
     - Save Column Mapping configuration
   - **No**

6. **Step 3: Enter generic Configuration Details**

7. **Step 4: Enter Schedule Configuration details and click Finish**

8. **Stop**
2. **Delete a job**

We can delete a job by passing its job name in the web-service. Quartz deletes the job from the job store and clears the information of the associated trigger and job from the quartz tables. Job deletion is irrevocable. All the information related to the job is lost.

3. **Pause a job**

A job can be paused by passing its job name in the web-service. Quartz pauses the trigger that is firing the job until it is resumed. Pausing a job does not cause any loss of information related to the job.

4. **Resume a job**

We can resume a job by passing its job name in the web-service. Quartz recalculates the next fire time of the trigger since its paused time and fires consequently.

5. **Re-run a failed job**

We can rerun by passing its ID in the web-service. The ID here refers to the Data Load ID. The ID is stored in the Quartz Job Data Map and this information is persisted during the job execution. If the Quartz Job Data Map contains an ID then it means that the job is supposed to be re-run, otherwise it is a normal job.

6. **Query job details of a particular job**

To view the details like start date, start time, end date, end time, repeat interval and status of a particular job we pass the job name in the web-service.

7. **Query all the job details of all jobs in the scheduler**
To view all the jobs and their details in the scheduler we design a web-service.

8. **Clear the scheduler**

This web-service clears all the jobs and associated triggers in the scheduler.

We present some of the sample RESTful Web-services in the Table 1 as follows:

Table 1

<table>
<thead>
<tr>
<th>Web-Operation</th>
<th>Sample Web-service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Job Scheduler</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/DeleteAll">http://localhost:8080/QuartzService/Jobs/DeleteAll</a></td>
</tr>
<tr>
<td>Create a job without start date, start time, end date, end time or repeat interval.</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Create/MyNewJob">http://localhost:8080/QuartzService/Jobs/Create/MyNewJob</a></td>
</tr>
<tr>
<td>Create a job with start date, start time and repeat interval</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Create/MyNewJob/20120329/2300/5">http://localhost:8080/QuartzService/Jobs/Create/MyNewJob/20120329/2300/5</a></td>
</tr>
<tr>
<td>Create a job with start date, start time, repeat interval, end date and end time</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Create/MyNewJob/20120329/2300/5/20120330/0900">http://localhost:8080/QuartzService/Jobs/Create/MyNewJob/20120329/2300/5/20120330/0900</a></td>
</tr>
<tr>
<td>Pause a job with its job name as URL parameter</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Stop/MyNewJob">http://localhost:8080/QuartzService/Jobs/Stop/MyNewJob</a></td>
</tr>
<tr>
<td>Resume a job with its job name as URL parameter</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Start/MyNewJob">http://localhost:8080/QuartzService/Jobs/Start/MyNewJob</a></td>
</tr>
<tr>
<td>Query a particular job with its job name as URL parameter</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Query/MyNewJob">http://localhost:8080/QuartzService/Jobs/Query/MyNewJob</a></td>
</tr>
</tbody>
</table>
4.3 Design of Rule Engine:

4.3.1 Design of database model for rule engine:

The rule engine is a part of the scheduling process responsible for evaluation of the data loaded by the scheduling system. Currently the rule engine supports rules for null check and range check. A probe can have any number of rules. In future, the rule engine is extendible to run rules in the form of scripts.

The data model for the rule engine consists of two tables namely RULES_CONFIG and RULES_AUDIT. The RULES_CONFIG table holds all the rules configured for a probe while the RULES_AUDIT saves the audit information of the rule. We design the RULES_CONFIG and RULES_AUDIT tables as shown below:

<table>
<thead>
<tr>
<th>Action</th>
<th>URL Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete a job with its job name as URL parameter</td>
<td><a href="http://localhost:8080/QuartzService/Jobs/Delete/MyNewJob">http://localhost:8080/QuartzService/Jobs/Delete/MyNewJob</a></td>
</tr>
<tr>
<td>Query all jobs currently inside the scheduler</td>
<td><a href="http://localhost:8080/QuartzService/Jobs">http://localhost:8080/QuartzService/Jobs</a></td>
</tr>
</tbody>
</table>

Create table RULES_CONFIG

(RULE_ID INTEGER NOT NULL,
RULE_NAME VARCHAR2 (50),
RULE_TYPE VARCHAR2 (255),
RULE_DESCRIPTION    VARCHAR2 (2000),
ATTRIBUTE_NAME       VARCHAR2 (100),
PROBE_NAME           VARCHAR2 (255),
MIN_VALUE            INTEGER,
MAX_VALUE            INTEGER,
RULE_SCRIPT          VARCHAR2 (4000),

CONSTRAINT RULES_CONFIG_PK PRIMARY KEY (RULE_ID)
);

Create table RULES_AUDIT
(RULE_LOAD_ID   INTEGER NOT NULL,
RULE_ID        INTEGER,
PROBE_NAME     VARCHAR2 (255),
RULE_EXEC_NOTES VARCHAR2 (2000),
RULE_EXEC_STATUS INTEGER,
RULE_EXEC_TIME  VARCHAR2 (50),

CONSTRAINT RULES_AUDIT_PK PRIMARY KEY (RULE_LOAD_ID),
CONSTRAINT RULES_AUDIT_FK FOREIGN KEY (RULE_ID)
REFERENCES RULES_CONFIG (RULE_ID)
);
RULE_ID in RULES_CONFIG table and RULE_LOAD_ID in RULES_AUDIT are Oracle sequences. If a rule is successful then the RULE_EXEC_STATUS is equal to one, otherwise zero.

4.3.2 Components of Rule Engine:

1. Knowledge Base for Rule Engine:

The basis for the Knowledge Base for the Rule Engine is the RULES_CONFIG database table behaving as a repository of rules. This table saves all the rules configured for each probe. We also use this Knowledge Base for reasoning to infer the conclusions.

2. Knowledge Acquisition Facility and User Interface:

We design a user interface for creation of rules while setting up a new probe. The user can also edit or remove rules from the Graphical User Interface. The user can also directly add rules for a probe from a database client by inserting the rules in the RULES_CONFIG table.

3. Explanation Facility for the Rule Engine:

We design an Explanation Facility to explain the reasons for success or failure of a rule for a probe. The source of this Explanation Facility is RULES_AUDIT table as it stores the audit information of each rule. Besides explaining the reason of failure for the rule,
the Explanation Facility contains other useful information like the fire time of the rule, rule id, and rule fire id.

4.4 User Interface Design:

We design the Web-based Graphical User Interface in a single web page. This page provides functionality to launch new jobs, re-submit failed jobs, edit probe configuration, monitor data load, pause, delete, unscheduled jobs, create rules, view rules audit etc. We divide this web page into two parts, top panel and bottom panel.

We dedicate the top panel to monitor job information, probe configuration. We create a grid to show the job information and two form panels to show the probe configuration. The first form panel contains the generic configuration details and the second form panel contains specific probe configuration details. We design a toolbar to hold various buttons to add, pause, delete, un-schedule a job, view rules of a job etc. and a filter bar also.

Figure 4 shows a screen-shot of the sample interface of the top panel:
The bottom panel contains a grid to show the information of data loads. We create a toolbar to hold buttons for re-run and a filter bar.

Figure 5 shows a screen-shot of the bottom panel:
4.5 System Implementation:

We build the service layer on top of Quartz and deploy it as Quartz Service inside JBoss Application Server. We build the Graphical User Interface separately and deploy this component inside JBoss Application Server. The server side code of user interface invokes the Web-service using the Remote Procedure Call. We use Groovy’s inbuilt XMLSlurper to perform this action. We pass the XML produced from the Web-service to XMLSlurper and parse the XML to separate the elements to feed the Graphical User Interface.

Figure 6 shows the complete picture to demonstrate the interaction between the front-end Graphical User Interface and back-end Scheduling System.
Figure 7: System Implementation
CHAPTER 5: RESULTS AND PERFORMANCE

5.1 Comparison Tests:

The most widely used scheduling system is UNIX cron. This scheduling system ships with all Unix-like Operating Systems. In this topic, we give a comparison between cron and Quartz. As FIT Scheduler uses Quartz Scheduling Framework, FIT Scheduler acquires all the properties of Quartz.

1. **JVM independent:** As cron is Operating System based, if the JVM (Java Virtual Machine) goes up or down, the state of the cron is lost. Though Quartz uses Java Virtual Machine for thread pools, Quartz maintains a separate mechanism to store trigger states.

2. **Multiple-thread scheduler:** Using FIT Scheduler, we can schedule multiple threads at a time. This is not possible with cron.

3. **Application Server support:** We cannot deploy cron in an Application Server. Hence, we cannot monitor the state of processes in a web page! However, FIT Scheduler is a deployable product. We can monitor processes, launch new ones etc. from the web page.

4. **Better portability of code:** Even though, cron is a utility in all Unix-like systems, the scheduling is not same in all the Operating Systems. As we use Java and
Groovy to build the scheduler, our code is highly portable as it runs on any JVM irrespective of its Operating System.

5. **Platform dependence:** Cron is platform dependent. Cron cannot run on machines other than Unix-like Operating Systems. Quartz can run on any Operating System.

6. **Application Integration:** Cron adds another entry-point to the application, while Quartz integrates into it. The developer need not bother about inter-process communication while using Quartz.

7. **Usability based on memory footprint:** If your tasks share nothing with each other, it is better to use cron. However, in a practical Enterprise Environment, a task shares a lot of information with other tasks. Using Quartz, we can create a process with as many threads as the tasks require.

8. **Other Enterprise-level Features:** Quartz has various Enterprise-level features such as Clustering, Load-balancing, and ability to make sure the successful completion of a task on server restart.

**5.2 Performance Analysis:**

In this topic, we present various performance outcomes of FIT Scheduler.

We compare the elapsed time to perform data load with data load ids and number of measurements queried for each data load to obtain the statistics. We restart the
Application Server and Database Server for each probe to avoid Oracle caching and Application Server caching.

Figure 8: Data Load ID vs. Elapsed Time for one Measurement Queried for each load
Figure 9: Data Load ID vs. Elapsed Time for four Measurement Queried for each load.

Figure 10: Measurements Queried vs. Elapsed Time in milliseconds.
From the above statistics, we understand that the performance of FIT Scheduler increases proportionally for each probe after each data load.
CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE

In this work, we approached the topic of building Enterprise-level Job Scheduling solutions from open-source projects. We also stressed on building layered architecture with lose-coupling between front-end and back-end to allow future scope extensively. We built Quartz Service before we designed the web-based front end to build GUIs following agile methodology.

The Rule Engine is also extensible in future. A future scope of development in the rule engine is to develop support to run rules in the form of Groovy or SQL scripts. Groovy’s inbuilt Groovy Script Engine and Groovy Shell are highly usable to evaluate the Groovy Rule Scripts. In the case of SQL Rule Scripts, we can design a query parser to evaluate and run the SQL scripts.

For performance evaluation, we have investigated various scenarios with different kinds of sources, spreadsheets as well as database tables. The outcomes of all the investigations proved to be successful besides increasing the performance proportionally after each data load. In future, we can extend the support of data sources to word documents, text documents etc. Besides this, we can add support for Email Jobs and Data back-up Jobs etc.
We built the user interface in Java using Google Web Toolkit 2.4.0 and Sencha’s JavaScript libraries (Ext-GWT 2.2.5) which are the latest versions during the time of development of this project. The support for these frameworks is highly extensive on their community forums.

We used Quartz 2.0.2 which is the most stable and latest version during the development of this project. Quartz releases versions for every couple of months fixing their bugs. For the next few versions of Quartz, we need not bother about the bugs as we have conducted various tests to evaluate the performance and durability of jobs, even during Daylight Saving Time.

Though we have used Oracle 11g R2 as our back-end database, FIT Scheduler is unaffected by its database platform as we used plain SQL to build the back-end. We have tested FIT Scheduler on Apache Derby 10.5.3.0 and MySQL 5.5.

Finally, we built a common framework to leverage various connection pools to perform the process of Extraction, Transformation and Data Load within a Servlet Container using Quartz Scheduling Framework and RESTful Web-services.
APPENDIX A.

JBOSS CONFIGURATION WITH QUARTZ

We install Quartz in JBoss Application Server even though we can embed Quartz libraries inside Quartz Service and deploy it. Following are the reasons for not embedding Quartz libraries inside Quartz Service deployment:

1. To make Quartz Service a lightweight application

2. To provide rights to change Quartz Scheduler properties to the administrators of the Application Server rather than the developers. If we embed Quartz libraries inside Quartz Service, we provide Quartz properties file along with Quartz libraries. We cannot change various properties on the fly like scheduler name, scheduler instance ID, thread count, job store’s data source etc. If we deploy Quartz properties separately from Quartz Service, the administrators can change these properties on the fly.

3. To create multiple scheduler instances dynamically

4. To provide security to the application

   Quartz properties file contains various important details like job store’s data source name and JNDI URL. If we embed Quartz properties inside Quartz Service, we expose this information to all the developers.

5. To make Quartz libraries available to other applications directly from Application Server rather than embedding Quartz libraries in each application.
All the versions of JBoss application servers ship with Quartz libraries. But the Quartz version (1.5.2) that ships with them is very old and it does not have the latest functionalities of Quartz framework. We use JBoss 5.1.0 which is one of the most stable versions from JBoss. We can download JBoss AS 5.1.0 from the official JBoss downloads page and select [jboss-5.1.0.GA-jdk6.zip] after redirecting to sourceforge.net.

Steps to install JBoss Application Server:

1. Unzip the downloaded zip file in a directory using WinZip or WinRAR or some other archiving utility.

2. Once you extract the archive, go to the bin directory and run the Windows Batch File run.bat. It takes a few seconds to start the server. You can open your browser and check http://localhost:8080/. This opens the home page of your application server.

Configuring JBoss Application Server with Quartz:

1. JBoss ships with many features. Some features are obstacles for Quartz Service while some are unwanted which slow down the Application Server. We remove those installations. Navigate to jboss-5.1.0.GA\server\default\deploy directory. This directory holds all the deployments including Quartz Service and FIT Scheduler. Files to be deleted are as follows:

b. Default data source file for HyperSQL - hsqldb-ds.xml

c. JBoss XA data sources file - jboss-xa-jdbc.rar. If we use two or more transactional resources in the same transaction, we must use xa-datasources. As we do not have such transactions we may delete this unwanted file from the deploy directory.

d. Java Messaging Service: jms-ra.rar. We do not use Java Messaging Service in our application. We may as well delete this factory-shipped file.

e. Factory shipped Quartz libraries - quartz-ra.rar. This is a very old version of Quartz. We install latest version of Quartz deleting this old version. Deleting this file is mandatory because of the class-loading conflicts.

f. Default schedule-manager-service.xml and scheduler-service.xml

g. Default Database exception service file - sqlexception-service.xml

2. Download and unzip the latest version of Quartz (2.x) from the [Quartz official web-site](http://quartz-scheduler.org). The most stable version of Quartz in 2.x is 2.0.2. We use the same version of Quartz to build Quartz Service.

3. Navigate to the JBoss installation directory ...\jboss-5.1.0.GA\server\default\lib. Add C3p0-0.9.1.1, Jta-1.1, Log4j-1.2.14, Quartz-2.0.2, Quartz-all, Quartz-backward-compat-2.0.2, Quartz-commonj-2.0.2, Quartz-jboss-2.0.2, and Quartz-
oracle-2.0.2 to the lib directory from the Quartz download. This installs the latest version of Quartz in JBoss AS.

4. Create data source for Quartz Job Store with the name QUARTZ_DS. Quartz Scheduler uses this DataSource to persist various details related to scheduler, jobs and triggers inside a database using JDBCJobStore mechanism inbuilt with Quartz. A sample for data source creation is shown below:

Create a file named oracle-ds.xml with the following content and deploy it inside the deploy directory of JBoss AS. We define other data sources in the same file.

```
<datasources>
  <no-tx-datasource>
    <jndi-name>QUARTZ_DS</jndi-name>
    <connection-url>jdbc oracle:thin:@localhost:1521:orcl</connection-url>
    <driver-class>oracle.jdbc.OracleDriver</driver-class>
    <user-name>scott</user-name>
    <password>tiger</password>
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter</exception-sorter-class-name>
    <min-pool-size>10</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <idle-timeout-minutes>1</idle-timeout-minutes>
  </no-tx-datasource>
```

5. Setup Quartz Database:

Navigate to the unzipped directory of Quartz download quartz-2.0.2\docs\dbTables. This directory contains database scripts for various database platforms like MySQL, Oracle, SQL Server, Sybase, Informix, HyperSQL etc.

We use Oracle as our database platform. Open the tables_oracle script and setup database tables in a schema (we use SCOTT schema).

6. Initialize logging system for Quartz. Navigate to ...\jboss-5.1.0.GA\server\default\conf directory. We can find a file named jboss-log4j.xml, which specifies the standard logging mechanism for JBoss. Logging is important to assure the successful deployment of Quartz properties file.

Navigate to jboss-log4j.xml file and add the following contents to the xml file. This automatically creates a log file under ...\jboss-5.1.0.GA\server\default\log with name quartz.log.

<!-- =================-->

<!-- Quartz -->

<!-- =================-->
Figure shows a sample screenshot of quartz-log after we deploy quartz-service.xml:
Figure 11: JBoss Standard Logging for Quartz Properties Deployment

7. Deploy Quartz Properties file inside JBoss AS as Managed Bean inside …\jboss-5.1.0.GA\server\default\deploy directory. We name the scheduler as “FitScheduler”. We provided a sample Quartz Properties File below:

Create a file named quartz-service.xml with the following content and deploy it inside the deploy directory of JBoss AS.
<server>

<mbean code="org.quartz.ee.jmx.jboss.QuartzService"

name="user:service=QuartzService,name=QuartzService">

<depends>jboss.jca:service=DataSourceBinding,name=QUARTZ_DS</depends>

<attribute name="JndiName">FitScheduler</attribute>

<attribute name="Properties">

    org.quartz.scheduler.instanceName = FitScheduler
    org.quartz.scheduler.instanceId = 1
    org.quartz.scheduler.rmi.export = false
    org.quartz.scheduler.rmi.proxy = false
    org.quartz.scheduler.xaTransacted = false
    org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
    org.quartz.threadPool.threadCount = 5
    org.quartz.threadPool.threadPriority = 5
    org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX
    org.quartz.jobStore.driverDelegateClass=
    org.quartz.impl.jdbcjobstore.oracle.OracleDelegate
    org.quartz.jobStore.tablePrefix = QRTZ_
    org.quartz.jobStore.dataSource = QuartzDS
    org.quartz.dataSource.QuartzDS.jndiURL = java:QUARTZ_DS

</attribute>

</mbean>
This completes the installation of JBoss AS and configuration with Quartz framework.
APPENDIX B.
QUARTZ SERVICE CODE MODULES

Setting-up the environment to build Quartz Service:

Software required for setting up the environment:

1. Eclipse IDE for Java EE Developers (Indigo)
2. Groovy plug-in for Eclipse
3. Apache POI – the Java API for Microsoft Documents Version 3.8 as it is the only version that supports Microsoft XLSX documents
4. Eclipse Explorer plug-in for interacting with databases during development.
5. Apache Ant Builder (This tool ships with Eclipse IDE for JEE Developers.)
6. Jersey implementation of JAX-RS
7. Oracle JDBC driver library (ojdbc6.jar)

Creating the Quartz Service project:

Our aim is to build Quartz Service as a deployable file and deploy in the JBoss AS. We can deploy Quartz Service as Web-archive (WAR file) or Enterprise-archive (EAR file) or Roshal Archive (RAR file) or in some other formats. We build our Quartz Service as a web-archive file named as “QuartzService.war”.

After installing the Groovy plug-in for Eclipse, we need to follow the below steps:
1. Create a new “Groovy Project” and name it QuartzService. This adds Groovy libraries and Groovy DSL (Domain Specific Language) libraries along with the standard JRE System Libraries.

2. Create a directory named lib and add all the quartz libraries and ojdbc6.jar to this directory. Add all these libraries to the project’s class path.

3. Create a directory named war and create a sub-directory named WEB-INF under war directory. Add web.xml and jboss-classloading.xml files.

**Web.xml:**

This file provides configuration and deployment descriptor file as an XML schema document. A sample Web.xml file is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"爱你?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>QuartzService</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
  </servlet>
  <init-param>
```

<param-name>com.sun.jersey.config.property.packages</param-name>
<param-value>com.caiso.fit.quartzService</param-value>
</init-param>
<init-param>
  <param-name>com.sun.jersey.config.feature.XmlRootElementProcessing</param-name>
  <param-value>true</param-value>
</init-param>
</servlet>

<servlet-mapping>
  <servlet-name>QuartzService</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>

**Jboss-classloading.xml**

To control the class-loading behavior of a war file we place this in the WEB-INF folder of our QuartzService application. A sample jboss-classloading.xml file is shown in Appendix [5] titled “Sample jboss-classloading.xml file”. The importance of jboss-classloading.xml file is in further topics.
<classloading xmlns="urn:jboss:classloading:1.0"
    name="QuartzService.war"
    parent-first="false"
    domain="DefaultDomain"
    top-level-classloader="true"
    parent-domain="Ignored"
    export-all="NON_EMPTY"
    import-all="true">
</classloading>

4. Under WEB-INF folder, create two directories named classes and lib. The "classes" directory holds all the compiled classes while the "lib" directory holds all the libraries that we intend to deploy in the server bundled inside QuartzService.war. Here lib directory contains all the Jersey libraries and Apache POI libraries (only the ones used for Spreadsheet loading). Add these libraries also to the project’s class path.

5. Create a build file to zip all the compiled classes under “classes” directory to a web-archive. A sample build file is shown in Appendix [7] titled as “Build file for Quartz Service”. We use Ant build tool to build the Quartz Service WAR file.

6. Create a local properties file named “local.properties” and leave it blank. You may need to use this if you want to setup any local properties for Quartz Service.
The project setup looks as shown in the following figure:
Figure 12: Project setup for Quartz Service.
Sample code snippets for web-services can be found in Appendix [8] titled as “Web-services on top of Quartz – Code snippets”.

**Getting context of “FitScheduler” programmatically inside RESTful web-service:**

We created a scheduler instance inside JBoss AS and named it FitScheduler. We need to get the context of this scheduler for scheduling the jobs. We pass this name to the constructor of javax.naming.InitialContext class in Java SE API to obtain the context to FitScheduler. This context is the scheduler object which can be used to perform all the operations like scheduling, unscheduling, rescheduling etc. A sample snippet is shown below:

```
StdScheduler scheduler = (StdScheduler) new InitialContext().lookup('FitScheduler').
```

1. **Web-services on top of Quartz – Code snippets:**

   a. **Create job only with name as parameter:**

```
@GET
@Path('Create/{JOBNAME}')
@Produces('application/xml')

public List<QuartzJobDetails> createWithoutTrigger(
@PathParam ('JOBNAME') String jobName) {

try {

    return new JobWithoutTrigger().create(jobName)

} catch (Throwable t) {

    return new JobWithoutTrigger().create(jobName)

}
```
b. Create job with start date, start time and repeat interval:

@GET @Path('Create/{JOBNAME}/{STARTDATE}/{STARTTIME}/
{REPEATINTERVAL}')</p>
@Produces('application/xml')

public List<QuartzJobDetails> createWithNoEndDate(

@PathParam ('JOBNAME') String jobName,
@PathParam ('STARTDATE') String textStartDate,
@PathParam ('STARTTIME') String textStartTime,
@PathParam ('REPEATINTERVAL') Integer repeatInterval,
@PathParam ('ENDDATE') String textEndDate = ' ',
@PathParam ('ENDTIME') String textEndTime = ' ')

try {
    return new JobWithoutEndDate().create(jobName, textStartDate, textStartTime,
repeatInterval, ' ', ' ')
} catch (Throwable t) {
    new QuartzServiceException().throwException(t)
}
c. Create job with start date, start time, repeat interval, end date and end time:

```java
@GET
@Path('Create/{JOBNAME}/{STARTDATE}/{STARTTIME}/
{REPEATINTERVAL}/{ENDDATE}/{ENDTIME}')
@Produces('application/xml')
public List<QuartzJobDetails> createWithEndDate(
    @PathParam ('JOBNAME') String jobName,
    @PathParam ('STARTDATE') String textStartDate,
    @PathParam ('STARTTIME') String textStartTime,
    @PathParam ('REPEATINTERVAL') Integer repeatInterval,
    @PathParam ('ENDDATE') String textEndDate,
    @PathParam ('ENDTIME') String textEndTime) {
    try {
        return new JobWithEndDate().create(jobName, textStartDate, textStartTime,
            repeatInterval, textEndDate, textEndTime)
    } catch (Throwable t) {
        new QuartzServiceException().throwException(t)
    }
}
```
d. Delete a job:

```java
@GET
@Path('Delete/{JOBNAME}')
@Produces ('text/plain')
public String delete(@PathParam('JOBNAME') String jobName) {
    try {
        return new Delete().deleteJob(jobName)
    } catch (Throwable t) {
        new QuartzServiceException().throwException(t)
    }
}
```

e. Pause a job:

```java
@GET
@Path('Stop/{JOBNAME}')
@Produces ('application/xml')
public List<QuartzJobDetails> pause(@PathParam('JOBNAME') String jobName) {
    try {
        return new Pause().pauseJob(jobName)
    } catch (Throwable t) {
        new QuartzServiceException().throwException(t)
    }
}
```
f. **Resume a job:**

```java
@GET
@Path('Start/{JOBNAME}')
@Produces ('application/xml')
public List<QuartzJobDetails> resume(@PathParam('JOBNAME') String jobName) {
    try {
        return new Resume().resumeJob(jobName)
    } catch (Throwable t) {
        new QuartzServiceException().throwException(t)
    }
}
```

g. **Re-run a job:**

```java
@GET
@Path('Rerun/{ID}')
@Produces('application/xml')
public List<QuartzJobDetails> rerun(@PathParam ('ID') Integer id) {
    try {
```
return new Rerun().rerunJob(id)
} catch (Throwable t) {
    new QuartzServiceException().throwException(t)
}

h. Query job details for a job:

@GET
@Path('Query/{JOBNAME}')
@Produces ('application/xml')
public List<QuartzJobDetails> queryJob(@PathParam('JOBNAME') String jobName) {
    try {
        return new QueryJobDetails().queryJob(jobName)
    } catch (Throwable t) {
        new QuartzServiceException().throwException(t)
    }
    
}

i. Query all jobs:

@Path('Jobs')
@GET
@Produces('application/xml')

public List<QuartzJobDetails> allJobs() {

    return new QueryAllJobs().getAllJobs()
}

j. Clear Quartz Scheduler:

@GET
@Path('DeleteAll')
@Produces ('application/xml')

public String clearAll() {
    StdScheduler scheduler=(StdScheduler)new InitialContext().lookup('FitScheduler')

    Log.info "Trying to Clear $scheduler.schedulerName..."

    scheduler.clear()

    Log.info "$scheduler.schedulerName has been cleared. All the Jobs and triggers associated with this scheduler have been deleted!"

    return "Scheduler - $scheduler.schedulerName has been cleared"
REFERENCES

1. The Apache License is a free software license authored by the Apache Software Foundation. It requires preservation of the copyright notice and the disclaimer.

   http://www.apache.org/licenses/LICENSE-2.0

2. The JBoss Application Server official downloads page

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

3. The Quartz Enterprise Job Scheduler official downloads page

   http://quartz-scheduler.org/downloads/catalog

4. Article on RESTful Web-services by Sameer Tyagi, dated August 2006

   http://www.oracle.com/technetwork/articles/javase/index-137171.html

5. Online Documentation of the JSR311 API (Java Specification Request) for JAX-RS (The Java API for RESTful Web-services), Version 1.1

   This project is under CDDL license. Copyright © 2009 Sun Microsystems, Inc.

   http://jsr311.java.net/nonav/releases/1.1/index.html


   Link to the official website of the open-source, production quality JAX-RS (The Java API for RESTful web-services) implementation and API for building RESTful (REpresentational State Architecture) Web-services

A free music website, which keeps a record of what, the subscribers listen to from any player. Based on the subscriber’s taste, the website offers recommendations for more music and concerts.


Link to the official page of JWatch, a browser-based, free, easy-to-use, open-source Quartz monitor to provide real-time monitoring to the Quartz Scheduling Framework.


Link to the official page of Citrine Scheduler, a Java web-application used to manage, configure and monitor various tasks (usually shell scripts).


Link to the official page of MySchedule project, a web-application for managing Quartz Schedulers besides providing extra components for easy programming.


Link to the official page of Apache POI – the Java API for Microsoft documents.


A timer-based job scheduler in Unix-like operating systems to enable the users to schedule jobs (usually shell scripts) to run periodically according to a given schedule, commonly used maintenance or administration system automation.
13. Effective Scheduling of Detached Rules in Active Databases by Stefano Ceri; Claudio Gennaro; Stefano Paraboschi; Giuseppe Serazzi published in IEEE Transactions on Knowledge & Data Engineering Jan/Feb2003, Vol. 15 Issue 1, p2 12p; 4 Black and White Photographs, 3 Diagrams, 1 Chart, 11 Graphs 10414347


19. A Legal Information Flow (LIF) Scheduler Based on Role-based Access Control Model by Tomoya Enokido; Valbona Barolli; Makoto Takizawa published in Computer Standards & Interfaces Sep 2009, Vol. 31 Issue 5, p906-912 7p 09205489

20. A Configurable Rete-OO Engine for Reasoning with Different Types of Imperfect Information by Davide Sottara; Paola Mello; Mark Proctor published in IEEE Transactions on Knowledge & Data Engineering Nov 2010, Vol. 22 Issue 11, p1535-1548 0p 10414347