Rooster Framework

Table of Contents

Chapter 1

Introducing Rooster

What can Rooster do for you? What is required to use Rooster? This chapter answers these questions.

Rooster in Brief

A framework streamlines application development by automating many of the patterns employed for a given purpose. It defines a set of functionalities which are common for any given application in the domain of the framework. It also adds good flow to the application, good structure to code, enabling the developer to write better,more readable and more maintainable code. Ultimately, a framework makes programming easier, since it packages complex operations into simple statements.

Rooster is a complete framework for developing resource scheduling applications using Genetic algorithms. It provides a solid mapping between resource constraint resource scheduling and genetic algorithms. Rooster framework also uses Genetic algorithms to optimize the generated schedule a level which can be defined by the user. Rooster framework's main focus is to generalize resource constraint scheduling optimization problem into optimization problem for Genetic algorithm and reuse of the existing functionalities to created applications for different domains . The end result of these advantages means there is no need to reinvent the wheel every time a new application is created.

Rooster is written entirely in Java. It has been thoroughly tested in , Rooster bank scheduling prototype application. Rooster Framework code base has been 90% covered with unit tests. It is compatible with most of the available databases engines, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. It runs on *nix and Windows platforms. Let’s begin with a closer look at its features.

Rooster Features

1. Easy to install and configure on most platforms (and guaranteed to work on standard *nix and Windows platforms).
2. Database engine-independent.
3. Easily configurable and easily extendable based on the application context.

Requirements of Rooster

To use Rooster in an customized application will require following.

1. JDK 1.6
2. Database engine of users preference.

Dependencies of Rooster

Rooster Framework's dependencies are as follows. While installation of Rooster these dependencies must be satisfied.

1. JGAP 3.6.1
2. Junit
3. Database connector for Java.

Rooster Team

Rooster Framework was developed as a final year project of University of Moratuwa department of Computer Science and Engineering, Sri Lanka. Project members are as follows.

1. Dr. A. Shehan Perera
   PhD (NDSU), MSc (NDSU), BSc (Hons) (Colombo)
   Senior Lecturer
   University of Moratuwa
   Computer Science & Engineering Department
   Sri Lanka.

2. Manorama Perera
   Undergraduate
   University of Moratuwa
   Computer Science & Engineering Department
   Sri Lanka.

3. Savithri Jayathilaka
   Undergraduate
   University of Moratuwa
   Computer Science & Engineering Department
   Sri Lanka.

4. Shanaka Kuruwita
Undergraduate
University of Moratuwa
Computer Science & Engineering Department
Sri Lanka.

5. Renukshan Saputhanthri
Undergraduate
University of Moratuwa
Computer Science & Engineering Department
Sri Lanka.

Rooster License

Fundamental Concepts

Java

Rooster Framework has been developed using Java programming language. Therefor solid understanding of Java and Object Oriented Programming is required to get the most out of the Rooster framework. Minimal version of Java development Kit required for Rooster framework is 1.6.

Object Oriented Programming(OOP)

The concepts of Object Oriented Programming will not be explained in this section. It is a vast spread area where explaining it will require a whole book it self. Since Rooster make extensive uses of the OOP mechanisms available as of Java. So it is essential to learn OOP concept as it is a prerequisite for Rooster Framework.

Java implements the object-oriented paradigms of class, object, method, inheritance, and much more. If the users are not familiar with OOP concepts, they are advised to read the related documentation.

Object-Relational Mapping (ORM)

Databases are relational and Java and most of other programming languages are object oriented. In order to access the database in an object-oriented way, an interface translating the object logic to the relational logic is required. This interface is called an object-relational mapping, or ORM.

An abstraction layer encapsulates the data logic. The rest of the application does not need to know about the SQL queries, and the SQL that accesses the database is easy to find. Developers who specialize in database programming also know clearly where to go.
Using objects instead of records, and classes instead of tables, has another benefit.

User can add new accessors to your tables. For instance, if you have a table called Client with two fields, FirstName and LastName, you might like to be able to require just a Full Name. In an object-oriented world, this is as easy as adding a new accessor method to the Client class,
like this:

```
public function getFullName()
{
        return $this->getFirstName().' '.$this->getLastName();
}
```

JGAP and Genetic Algorithms

Genetic algorithms are adaptive methods used to find optimum solutions for problems. It is an evolutionary algorithm. According to the categorization mentioned early, Genetic Algorithms is a stochastic search method. The main principle behind GA is base on survival of fittest. The evolutions over many generations should result in surviving only the fittest features included in the chromosomes.

JGAP is a Java framework for Genetic algorithms. It provides basic genetic mechanisms that can be easily used to apply evolutionary principles to problem solutions. JGAP comes with following features.

1. Implement the fitness function
   Users can extend this function to define their own fitness function.
2. Setup a Configuration object
   GAP is very flexible and pluggable. For example we can create our own genetic operators,can have random number generators and can do natural

selections etc. For these things JGAP uses a Configuration object. There is also a DefaultConfiguration class that comes with common settings.

3. Create a population of potential solutions.
   Population of Chromosomes is called a Genotype. Genotype class is used to create the population. There are two main options to create population. Construct each Chromosome individually and then pass them all into a new Genotype and creating a random population are those two options.

4. Evolve the population
   We can simply call population.evolve () to evolve the population. If need to check the bestchromosome which is the most fittest one among the population after each iteration we have to invoke the getFittestChromosome() method on the population

More details of JGAP can be found in JGAP Framework documentation.

Summary

Rooster is a Java and Genetic algorithms based framework for resource constraint resource scheduling and optimization. It uses basic concepts in programming like OOP, ORM and design patterns. Rooster framework can be used to develop resource scheduling applications in different domains. It contains basic but necessary functionalities to use genetic algorithms in the resource constraint resource scheduling.

Chapter 2

Exploring Rooster

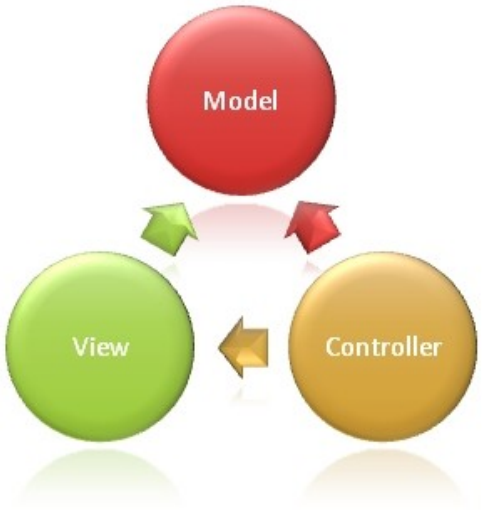
This Documentation will give the user an overview of the inside of Rooster Framework. Basically Architecture behind the Rooster Framework, Code structure , Configuring the Rooster Framework, How to develop an application using Rooster Framework is the main topics discussed in this Rooster Framework guide document.

Rooster Framework Architecture

Rooster Framework has been Architectured based on well established architectural pattern called MVC. Main reasons for using MVC pattern is to

1. Isolation of business logic from the user interface resulting in greater maintainability .
2. Isolating the data manipulation.

MVC contain three basic layers, namely Model layer, Control layer , View layer. Model layer represents the data on which the application operates. View renders model into a form suitable for interaction. Controller responds to the user request and invoke changes into model or view.



Resource : http://www.w3schools.com/aspnet/mvc_intro.asp

MVC in Rooster

1. Database Abstraction

Model layer can be viewed as a collection of database access layer and data abstraction layer. Data access layer will manage the data access operations. Data access object can be used to retrieve data effectively.

Data access functions will not use any database dependent queries and data abstraction layer will handle the database dependent queries. That way application get more flexibility when migrating from one database to another. When migrating from one database to another database we only have change the Data Abstraction layer. We can reuse the data access layer.

2. Control Abstraction

Control layer is also modularized to easily access and maintain the control logic isolated from the model and view layers.

Chapter 3

Installing Rooster

This Chapter will give an introduction to how to install and configure Rooster framework. After a successful installation user will be able extend the framework to created scheduling applications.

Prerequisites

Before installing Rooster, you need to check that your computer has everything installed and configured correctly. Take the time to conscientiously read this chapter and follow all the steps required to check your configuration, as it may save your day further down the road.

Java JDK 1.6 installation

For developing Java applications its essential to have Java JDK platform installed. More details regarding the installation can be found in
http://www.oracle.com/technetwork/java/javase/index-137561.html

Database Engine Installation

Since Rooster Framework depends on a database to store initial data and scheduled data, its mandatory to have a database engine installed in the system. User/Developer can select a database to their liking and initial data should be loaded accordingly.

JGAP

Since Rooster Framework depends on JGAP library its essential to have JGAP installed and configured correctly so that Rooster framework can use the functionalities of JGAP properly. JGAP installation can be found in
http://jgap.sourceforge.net/doc/install.html

Rooster Installation

After successfully completing prerequisite installation in to the system, user should proceed with the installation of the Rooster .

Step 1

Download Rooster Framework from Rooster web-site. If user is planing to improve the functionalities in Rooster framework it self its advice to get a check-out from following link

https://rooster.googlecode.com/svn/trunk

Step 2

Verify the dependencies of Rooster is satisfied. Create the test database and load default data into the test database.

Step3

Run the unit tests and verify them. In case of a unit test failure user will have to identify the error and fix it. Proceeding with the application with out fixing it can lead to errors in the application that user developing.

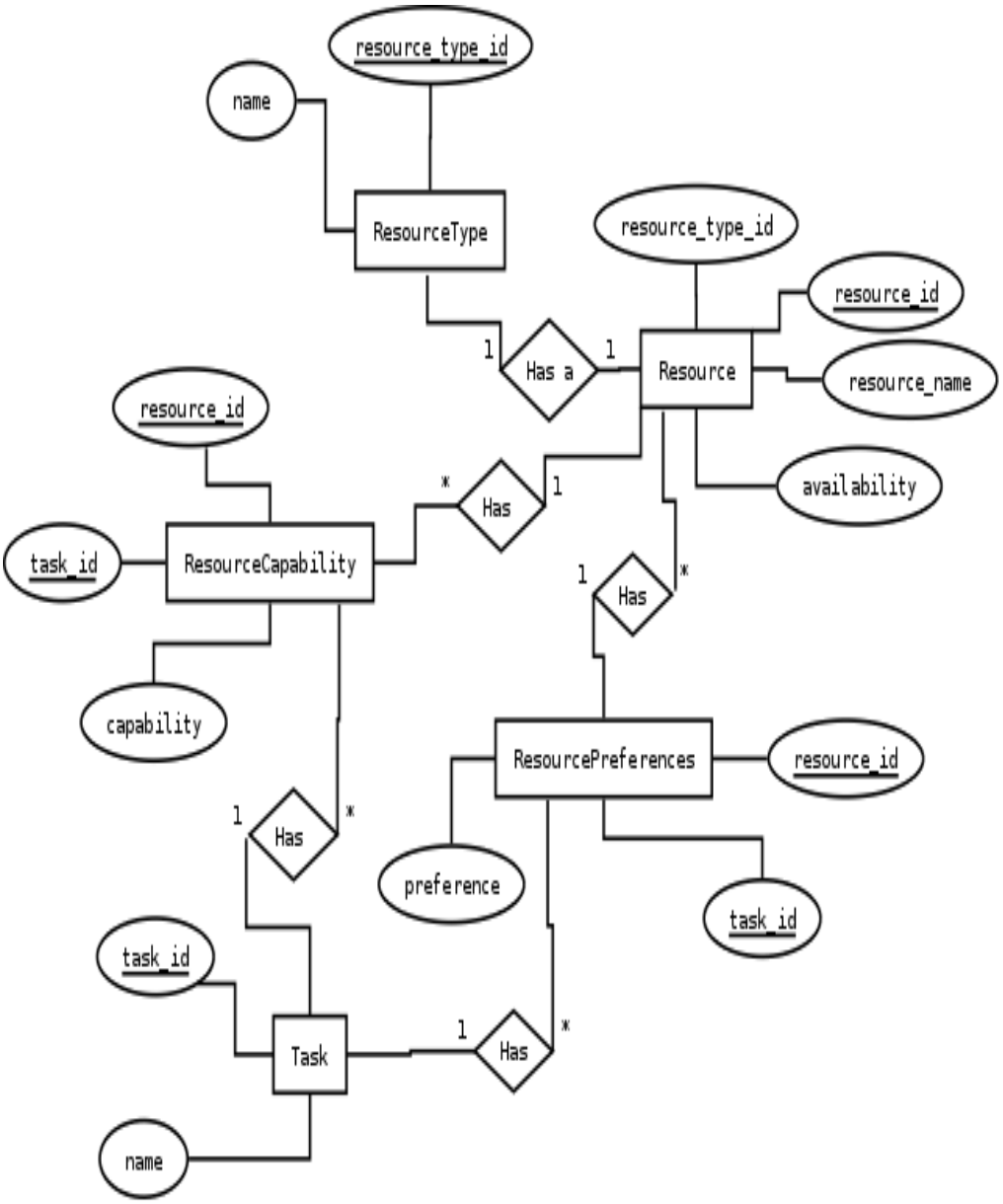
Chapter 4

Model Layer

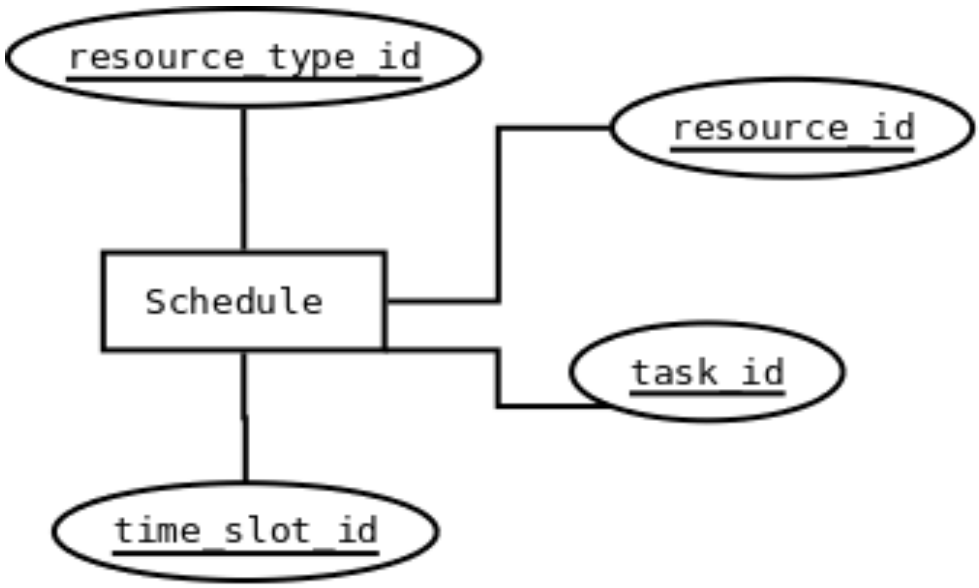
In model of the Rooster Framework basic data logic which requires to operate scheduling is handled. Rooster Framework use MySQL as the Framework's database and uses Java MySQL database connector to establish the database connection with database.

Rooster Database Tables

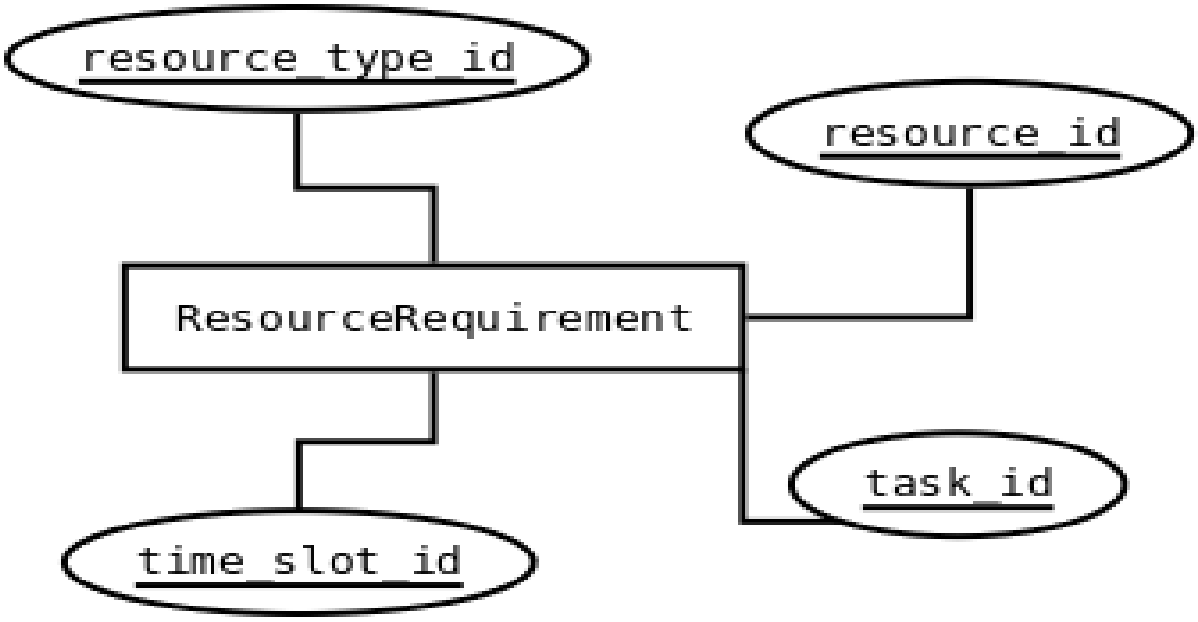
Table Name	Description
rooster_resource	Stores information about resources
rooster_resource_capability	Stores information about resources capability
rooster_resource_preferences	Stores information about resources preferences
rooster_resource_requirement	Stores information about resources requirement for the schedule to be optimized
rooster_schedule	Stores information about created schedule
rooster_task	Stores information about tasks
rooster_time_slot	Stores information about time slots
rooster_resource_type	Stores information about resource types ex : volume,non volume, printer etc



Resource related ER Diagram of the Rooster Framework



Schedule Table ER Diagram of Rooster Framework



resource Requirement Table ER Diagram of Rooster Framework

Use of ORM in Rooster

Databases are Relational. Java is Object Oriented. In order to access database effectively in object oriented context proper mapping, an Interface translating the object logic to the relational logic is required . This Interface is called ORM. Main advantage of using ORM is re-usability allowing the methods of a data object to be called from various parts of the application, even from different applications. ORM also encapsulates the data logic to objects.

1. RoosterRepository.java

Rooster Framework uses ORM to achieve above defined benefits of the ORM. Rooster ORM can be found in org.rooster.dbmanager.repository RoosterRepository.java. This class contains set of wrapper functions ,which uses a particular DataSource class which is initialize and created based of the configuration stored in a database.properties file.

2. DataSource.java

This is an Interface class which all the DataSource classed used in the RoosterRepository class should implement. This interface contains predefined data access methods to be implemented based on the database type to be used in the Rooster application.

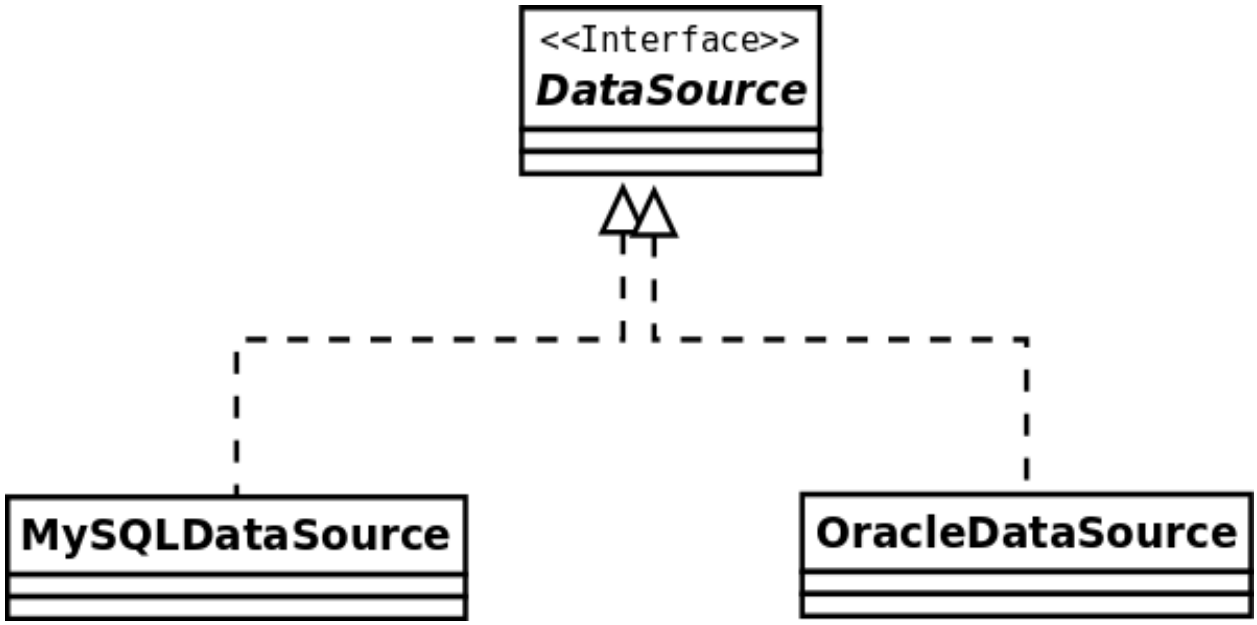
3. RoosterDataSourceFactory.java

Rooster Framework used Factory pattern to create the required Data source class based on the database.properties file.

4. MySQLDataSource.java

Since Rooster Framework comes with a default MySQL based database, cooresponding data source file is already created by Rooster Framework management team. This class implements the Interface Datasource.java class.

Rooster Data source Class Diagram

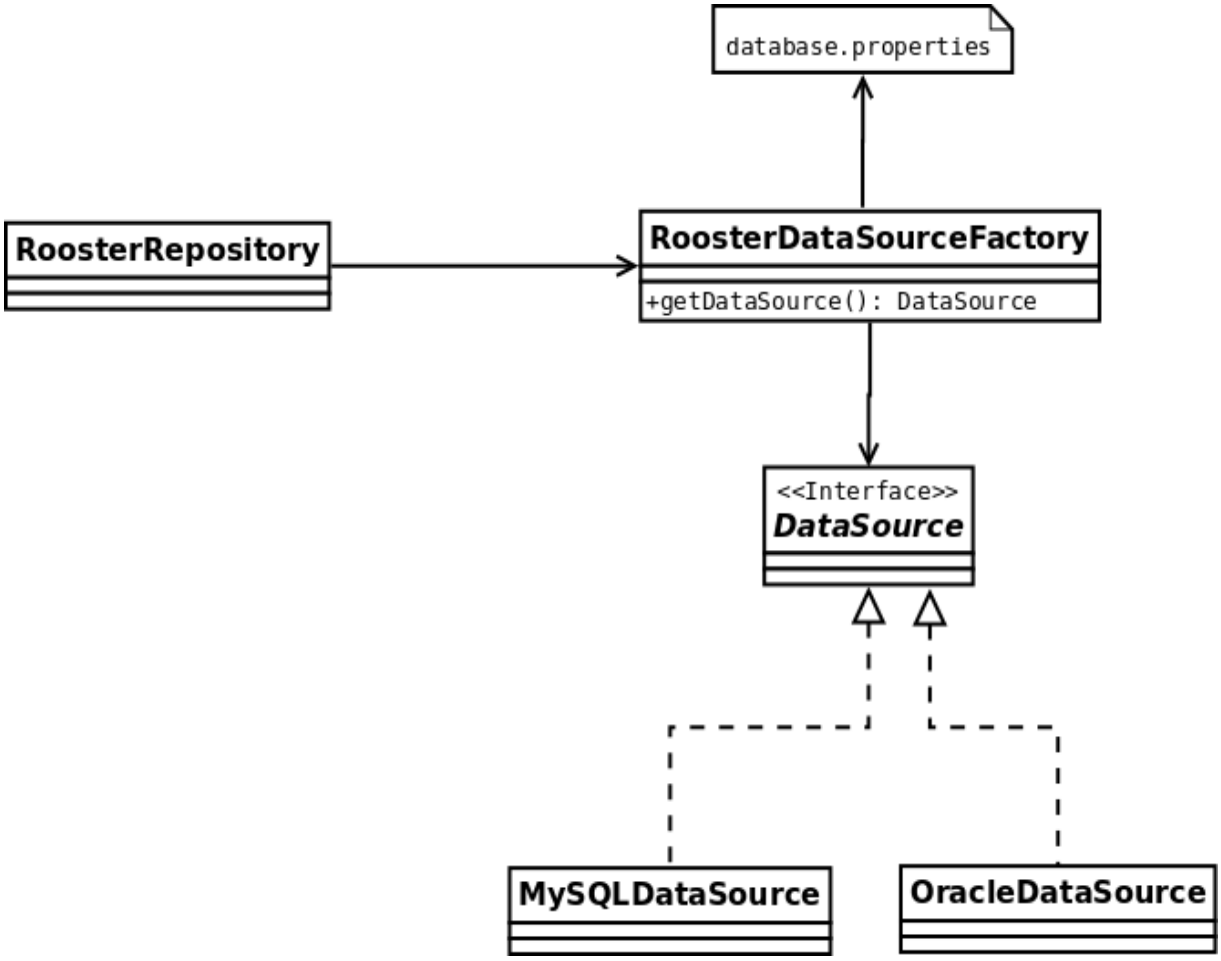


Factory Pattern

Factory pattern is used to support different representations of some attribute depending on the some external factors. Factory pattern is used to encapsulate a group of individual attributes which has common theme. This pattern separates the details of implementation of a set of objects from their general usage.More detail regarding the factory pattern can be found

http://en.wikipedia.org/wiki/Abstract_factory_pattern

Rooster Data source Creation using Factory Pattern



Chapter 5

Control Layer

The controller layer, which contains the code linking the business logic and the view, is split into several components that you use for different purposes .

1. The controller provides the unique entry point to the application. It loads the configuration and determines the resource requirement and configure the application according to the view layer input.
2. The controller layer prepare the data needed by the presentation layer .
3. Controller determines which constraints to be apply when generating the schedule.
4. The controller provides a set of termination conditions which is used to terminate the application.
5. Controller encodes the Chromosome based on the schedule requirement and decodes the generating schedule and output that to the presentation layer.
6. Controller performs the Genetic operations and evolve the population generated based on the encoding.

Controller as an Encoder

Controller is responsible for encoding the resource, tasks and time slots based on the requirement of the schedule into Genes which are used to generate the Chromosome. Then Chromosome is used to evolution and and it will be evaluated against the constraints and rules defined by the configuration of the application.

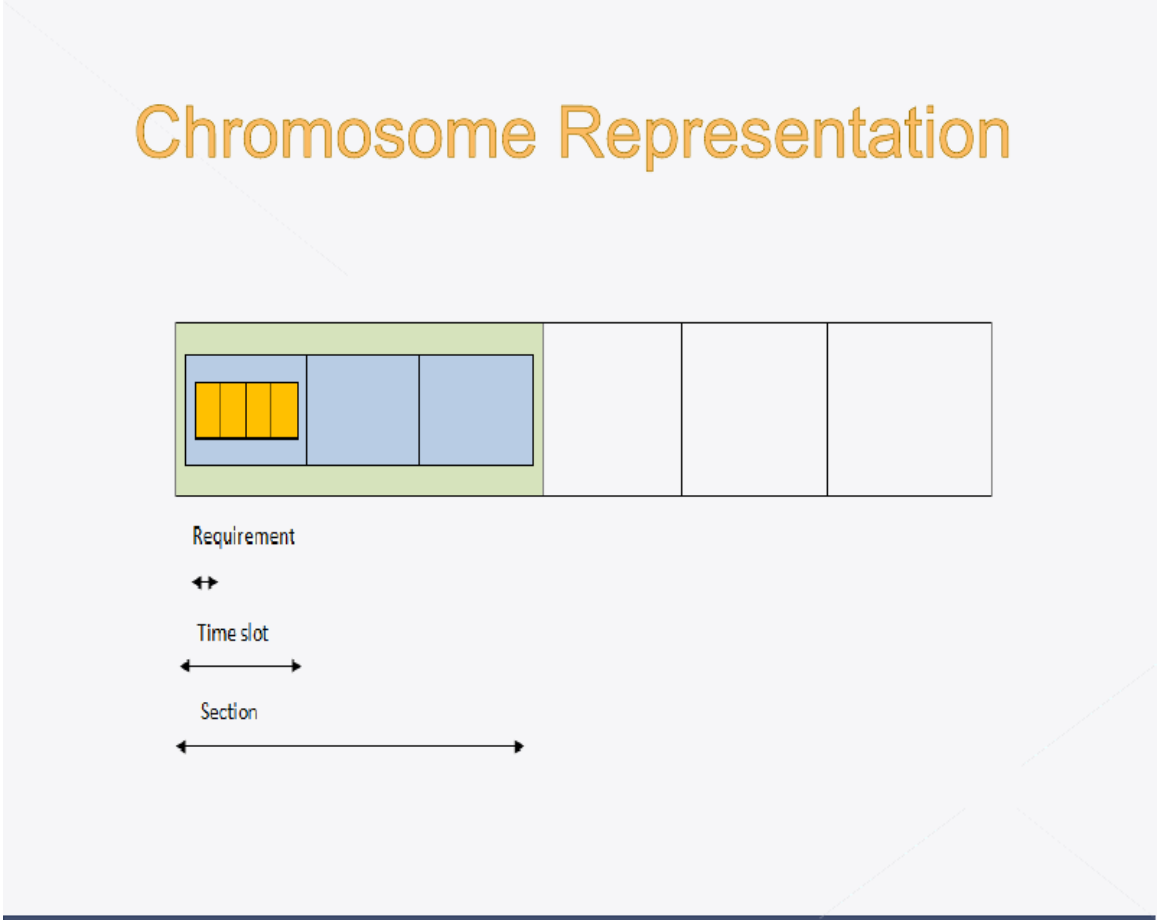
In Rooster framework org.rooster.encoder package is responsible for encoding the requirement to Genes and then to Chromosome. This is one of the important functions of the application since the accuracy of the generated schedule depends on the encoding used to evolve the chromosome.

1. Encoder.java class

Encoding logic is captured in this class. This class calls to the ORM layer and retrieves the resources stored in the database and maps each type of resources to Hash maps. Then it provides Encoding logic based on the Maps generated by the Encoder. Resource mapping is initialized every time when the application encoding the resource requirement into Genes.

Since Rooster uses Integer genes in the application, resource scheduling problem should be mapped into Integer genes. Integer Genes class takes two parameters as int lower bound and int upper bound. These parameters defines the upper and lower bounds of a particular resource type. These parameters are then again used in the JGAPMain.java class while initializing the sample gene array and then it is used to define the Chromosome.

After generating the schedule we must decode the fittest Chromosome's genes based on the encoding logic that we have used to encode. Decoding is also done in the Encoder.java class.



Controller as Genetic operations performer

Rooster Framework has modularized the Genetic operations which should be performed to obtain the optimal and efficient solution for a given scheduling problem. These modularization is important to easily extend the framework depending on the customized requirements.

1. JGAP configuration management module
2. Rooster genetic mapper module
3. Optimization monitoring module

1. JGAP configuration management module

1.1 JGAPMain.java class

Main functionality of the controller is to perform and handle genetic operations on the encoded Chromosome population and optimize the schedule. Population needs to evolved by doing genetic operations like crossover and mutation. In Rooster framework's implementation JGAPMain.java class is responsible for performing all the genetic operations with the use of JGAP library. JGAPMain.java class is runnable class, it not only performs genetic operations on the encoded population but also it sets up the genetic algorithm optimization default configuration settings. It initialize the Fitness function by embedding the Rules in other words the constraints which should be applied to the evaluate the fitness of the Chromosome.

Listing 2.1 Setting the default configuration in JGAPMain

```
Configuration defaultConfiguration = new DefaultConfiguration();
setMutationRate(defaultConfiguration);
FitnessFunction scheduleFitness = new JGAPFitnessFunctionImpl(ruleEvaluator);
defaultConfiguration.setFitnessFunction(scheduleFitness);
```

Listing 2.2 Setting the Chromosome in JGAPMain.java

```
Chromosome sampleChromosome = new Chromosome(defaultConfiguration, sampleGenes);
defaultConfiguration.setSampleChromosome(sampleChromosome);
defaultConfiguration.setPopulationSize(POPULATION_SIZE);
Genotype population = Genotype.randomInitialGenotype(defaultConfiguration);
```

After setting up the default configurations which is needed to run the optimization JGAPMain.java class identifies the termination conditions which are used to terminate the evolution of the population. Currently in Rooster framework have three per-implemented termination conditions. If particular user to included another termination condition, it should be added to JGAPMain.java.

Logic of the Termination should be defined as class to the org.rooster.genetic.monitor package and should be properly handled according to the semantics of the application.

Listing 2.3 Setting the termination in JGAPMain

```
if (timeMonitor) {
   evolveUntilMaxTimeLimitReach(population, MAX_TIME);
} else if (fitnessMonitor) {
   evolveUntilRequiredFitness(population, MAX_FITNESS);
} else if (progressWithTimeMonitor) {
   evolveUntilThereIsNoProgressWithinTime(population, MAX_TIME);
} else if (progressWithIterationsMonitor) {
   evolveUntilThereIsNoProgressWithinIterations(population, NO_OF_ITERATIONS);
}
```

2. Rooster Genetic Mapper Module

JGAP library has been implemented using java. Then users might think what is the need of mapper since both sides are Object oriented . Since Rooster Framework's main objective is to provide set of functionality which can be easily extended to address any type of resource constraint scheduling problem, its required to map the Genes, Gene indexes etc in the Genetic side to more Generic resource scheduling attributes like resource amount, resource type etc.

Rooster provides set of generic mappers in org.rooster.genetic package. These mapper are used to transfer data from one layer to another, perform operations on each layer effectively etc.

2.1 GeneIndex.java

Rooster GeneIndex class is the mapper class between the Genetic algorithm's Gene class and the schedule's task,time slot and resource type.

This can be categorize as data mapper class. Since we have encoded our resource requirement into Genes, to perform various actions such as evaluating the fitness against a set of rules we need a this mapper to map Genes attributes with the schedule's attributes and vise versa. All rule implementations provided in the Rooster Framework uses this mapper to perform fitness evaluation.

Listing 2.4 Example mapper function

```
public static int getTimeSlotStart(int sectionNo, int timeSlotNo) {
    int timeSlotStart = 0;
    timeSlotStart += getSectionStart(sectionNo);
    for(int i=0; i<timeSlotNo; i++) {
    timeSlotStart += TaskManager.getRequirement(sectionNo, i);
   }
    return timeSlotStart;
}
```

2.2 Interface GenericRule

GeneIndex class described above is data mapper. It basically maps data between Genetic attributes and schedule's attributes. But there can be many mapper variations other than data mappers class GenericRule provide a an Interface to add scheduling constraints to the Genetic algorithm's engine.

User's can extend GenericRule interface and define rules which are to be evaluated against the generated Chromosomes. GenericRule class provides a well defined set of functionalities to interact with the Genetic algorithm's functionalities. So this class can also be defined as a functionality mapper.

Listing 2.5 A Method defined in GenericRule class

```
public double fitnessValueForRule(IChromosome chromosome);

public int getWeight();

public void setWeight(int value);

public String getName();
```

2.3 JGAPFitnessFunctionImplementation Class

This class is used in the Rooster Framework to implement the fitness function class in JGAP library which should be overridden based on the context of the application. Since Rooster Framework's main goal is to provide a context independent scheduling Framework , evaluating fitness of a particular Chromosome is configurable. Fitness calculation logic is pushed into separate rule classes which can be added to the JGAPFitnessFunctionImplementation class to calculate the total fitness of the Chromosome against the user defined rules. Main requirement of this class is to map the differences between the scheduling context based fitness function implementations into a general interface which user can use.

Listing 2.6 Overridding the evaluate method

```
   @Override
protected double evaluate( final IChromosome chromosome ){
  ScheduleMetrix.getScheduleMetrixInstance().fillMetrix(chromosome);
  double fitness = 1.0;
  for(int i=0; i<rulesArray.length; i++) {
    fitness *= Math.pow(rulesArray[i].fitnessValueForRule(chromosome),
          rulesArray[i].getWeight());
 }
           return fitness;
}
```

3. Optimization Monitoring Module

Resource scheduling problems falls into NP-Complete problem space. So the level of optimization is an important factor when using optimization algorithm. The optimization of the Genetic algorithms depends on various factors. But in Rooster Framework,the level of optimization of genetic algorithms depends of the three predefined factors.

1. Progress with Iterations
2. Progress with Time
3. Time

These also can be viewed as termination conditions for the Genetic algorithm. But these conditions should be monitored and controlled. Since the accuracy of the schedule depends on these.

Different context might depend on the different optimization levels and optimization factors based on the context itself. org.rooster.monitor packge allows the users to use the existing termination conditions defined by the Rooster Framework or users can define the termination conditions based on the attributes of the context that the application is been used. To add a different termination condition,
users must come up with a termination logic and embed that logic into Rooster application by adding it as a class into org.rooster.monitor package.

Listing 2.7 Example method and constructor in ProgressWithIterationsMonitor

```
public ProgressWithIterationsMonitor(Vector<Double> vector, int noOfIterations) {
  this.fitnessValueVector = vector;
  this.NO_OF_ITERATIONS = noOfIterations;
 }

public boolean isMAX_PROGRESS_REACHED() {
   return MAX_PROGRESS_REACHED;
  }
```

Controller as a Terminator

Rooster controller handles the termination of the optimization of the Genetic algorithms. Proper termination conditions should be provided based on the context of the application and termination should be handled in such a way that it does not affect the optimized solution. Rooster Framework provides three predefined termination conditions.

1. Number Progress with Iterations
2. Progress with Time
3. Time taken to Optimization

Many other termination conditions can be introduced based on the context as well as the user requirement. Termination conditions should be added under org.rooster.monitor package. Then selected termination condition should be passed JGAPMain.java class where evolution of the Chromosomes is managed.

Controller as a Constraint Manager

Rooster Framework is been developed mainly to address issues encountered while resource constraint scheduling. So adding constraints and maintaining the relationship with the constraints is an important aspect of the Rooster Framework. If we have the constraints in the coded level and hard-code the flow of the constraints without using a configurable constraint approach, adding a a new rule will have to be done by doing some core function changes and attribute changes. This make the code corresponding to adding rules more brittle and vulnerable.

If we include our constraint logic into code and introduce a predefined flow, adding a new constraint will break the follow and that can be resulted in creating the more practical schedule not the Optimal effective schedule.

Solution provided in Rooster Framework is to remove the brittle code from the application by capturing these logic into different classes and embedding them in the run time based on the users configuration settings. The configuration setting can be provided by the View layer or it can be derived based on a configuration file stored in the database.

Controller itself is responsible for managing the constraints. Control layer provides GenericRule Interface to implement for any rule which should be applied to the scheduling process. Basic components of constraints in controller can be view as

1. Defining the Rules(Constraints).
2. Embedding the Rules in to the application

1. Defining the Rules

org.rooster.chromosomeevaluator.rules package is used to defined the Rules which are to be applied into the application. Rooster Framework has predefine generic rules set which is defined under the org.rooster.chromosomeevaluator.rules package. If a particular user requires different rule which should be added in to the system, that rule should be defined inside this package.

These rules should implement GenericRule Interface in oder to have a mapping section to the system. Rules should override the fitnessValueForRule method of GenericRule of the Interface based on the logic of the rule and should return end fitness values for the Chromosome which the rule is validated against.

Listing 2.8 fitnessValueForRule method for EmployeeCapabilityCheckingPenaltyRule

```
 public double fitnessValueForRule(IChromosome chromosome) {

        int penalty = 1;
        int sectionStartIndex;
        int sectionEndIndex;
Gene[] genes = chromosome.getGenes();
  for(int i=0; i< TaskManager.NO_OF_SECTIONS; i++) {
          sectionStartIndex = GeneIndex.getSectionStart(i);
          sectionEndIndex = GeneIndex.getSectionEnd(i);
          for (int j=sectionStartIndex; j<=sectionEndIndex; j++) {
           if(ResourceCapabilitiesTracker.getInstance().getEmployeeCapability((Integer)
                genes[j].getAllele(), i) == 0){
                 penalty++;
                 incapableAssignments.add(new int[]{(Integer)genes[j].getAllele()-1,i});
                }
          }
          }
                 return 1.0/(penalty * weight);
}
```

2. Embedding the Rules in to the application

To embed a defined rule into the Rooster Framework, RuleEvaluator class which is under the org.rooster.chromosomeevaluator is used. RuleEvaluator will maintain a HashSet of GenericRule class to add each user defined rules into the System. This HashSet is then passed to the FitnesssFuntionImplemetation class to calculate the total fitness. Since Rooster code does not know the rule flow logic (hard coded) we can add as many as rules without converging into a practical solution or having volatile code.

Listing 2.9 example methods of RuleEvaluator

```
public void addRule(GenericRule genericRule) {
   ruleSet.add(genericRule);
}
```

```
public int getNoOfRules() {
   return ruleSet.size();
}

public GenericRule[] getRulesArray() {
   int i=0;
    GenericRule[] rulesArray = new GenericRule[ruleSet.size()];
 Iterator iterator = ruleSet.iterator();
 while (iterator.hasNext()) {
              rulesArray[i] = (GenericRule) iterator.next();
             i++;
          }
             return rulesArray;
}
```

Controller as a Data/Functionality provider to View/User

Controllers is responsible for preparing data required for the view layer to function. View layer should use the functions defined in the control layer to full fill its data requirements. Rooster Framework provides a set of well defined functions to end user to define the view layer of the Rooster application.

When extending the Rooster Framework and creating a customized application users might need some of the functionalities and data that has been kept inside the Rooster Framework. Rooster org.rooster.service package provide a Service Layer inside the Control layer to address the specific and general user requirements. Service.java class is responsible for exposing set of functionalities which view requires to generate the view layer.

1. Service.java Class

For a application developed using Rooster Framework, it is must to communicate and store the user input from the UI. Rooster Service.java class exposes most of the generic configuration saving methods and the data retrieval methods to retrieve schedule's data from the database. Service.java class can also be used as a entry point to the Rooster Framework. By which users can extend the Rooster Framework.

Listing 2.10 example methods of Service

```
 public void setTerminationParamerts(boolean fitnessSelected, int fitness, boolean
          timeSelected, int time, boolean progressWithinTimeSelected, boolean
progressWithinIterationsSelected, int noOfIterations) {

   JGAPMain jgapMain = new JGAPMain();
  jgapMain.setParameters(fitnessSelected, fitness, timeSelected, time,
progressWithinTimeSelected, progressWithinIterationsSelected, noOfIterations);
            }

public void setResourceCapability(ResourceCapability resourceCapability) {

 RoosterRepository.getInstance().updateResourceCapability(resourceCapability);

 }
```

Chapter 6

View Layer

Rooster Framework provide the flexibility to the users to decide the whole View layer of the application. This includes the technology that user prefer to develop the UI, features that are included in the UI, look and feel of the UI for the end user etc. Rooster Framework does not provides users with a built in UI. Hence users who are engine to generate schedule must develop own UI based on the requirements.

Rooster Framework provides most of the required data access and the functionalities in the Service class which can be found inside the org.rooster.service package.

Reasons for Not Implementing Customized View Layer

1. Remove the UI technology dependency in Rooster application .
   - User can use their preferred UI technology.
2. Application can be used in different context and sectors
3. Since Rooster is Generic application.

Chapter 7

Reliability of Rooster and Testing

Reliability of Rooster

1. Rooster has been tested with a prototype application for generate banking schedules. So Rooster team is confident that Rooster Framework is Reliable to use.
2. Rooster Framework has been developed using well known and established software development methodologies.
3. Rooster Framework maintains a good code quality and uses coding best practices.
4. Rooster Framework is under open source license. Hence it will be evolving and maintained properly
5. Rooster Framework will have over 90% of code coverage via unit tests.

Unit tests

Rooster framework's methods are covered using unit tests. Other than the view layer which I user have to implement, Model and Control layers are covered with unit test which validates the method. For writing unit tests, Rooster uses JUnit framework.

Test classes can be found in Test Packages. Model and View layers of the Rooster has more that 90% line coverage for each class. Rooster uses cobertura plugin for generating code coverage reports.

GNU Free Documentation License

Please refer following link to get the license of this document.

http://www.gnu.org/licenses/#FDL