FEATURE ORIENTED SOFTWARE DEVELOPMENT METHODOLOGY FOR STOCK EXCHANGE SYSTEMS

Lasitha Harinda Konara

(168235T)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 2018

FEATURE ORIENTED SOFTWARE DEVELOPMENT METHODOLOGY FOR STOCK EXCHANGE SYSTEMS

Lasitha Harinda Konara

(168235T)

Thesis submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 208

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ……………… Date: ………………..

Name: K.M.G.L.H. Konara (168235T)

The above candidate has carried out research for the Masters Dissertation under my supervision.

Name of the supervisor: Dr. Indika Perera

Signature of the supervisor: …………………………. Date: ………………..

Abstract

Many organizations that develop software use the traditional method of layered methodologies to develop their end software product or solution. In doing so, the code will be a more general one and there will be a lot of unnecessary elements included which make the system heavy and dirty. This would result in a lot of issues .Also there is a requirement to implement a system with a concept of features. The end system will be delivered as a set of features and the feature set could be decoupled at any time, according to the current requirement without harming any existing functionality.

This research has been narrowed down to a particular domain which is the stock exchange or trading domain. By narrowing down the domain, the end software product could be delivered in a tailor made manner so that its effectiveness will be very high. The final software product would be a feature oriented domain specific language (DSL).

The objective of the feature oriented DSL is to make it very effective even for business analysts to introduce new features without getting help from core software developers. The feature layer will be purely decoupled and presented in an independent way so that the end users will have full flexibility to introduce changes very easily. There is a clear separation between core code segments and auto generated code segments. Auto generated files serve the different features and core code segments will enable those features to function on top of them. Auto generated code should not be changed manually under any circumstance as per this design.

A code generator and the core controller is developed throughout this research exhibiting the above mentioned feature oriented software development principles and domain specific language principles.

Keywords : FOSD, DSL,FOP, AOP, ANTLR,Entity, Instance

ACKNOWLEDGEMENT

My heartfelt gratitude is given first and foremost to the academic supervisor of this research Dr.Indika Perera for his immense help, support and advice given throughout the course of this research. His encouragement and contribution in the form of alternative methods, ideas and concepts provided motivation for us to move forward with this research.

My special thanks goes out to my external research supervisors/advisors, Mr.Manoj Bandara, Mr.Sujith Gunawardhane, Mr.Surith Pinto and Mr.Sampath Thilakumara for giving me the opportunity to work on this research, providing the guidance .Also I would like to thank all the staff members in MillenniumIT Software (Pvt) Ltd.

Academic staff members' valuable guidance and advice since the very beginning of this research seminar series is also highly appreciated. Specially I should appreciate the support and advice given by Dr. Malaka Walpola for encouraging us to do research.

My gratitude is also extended to all the staff members of the Computer Science and Engineering Department who provided us with numerous advice and feedback, especially at reviews, feasibility study. I appreciate your guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Description
FOSD	Feature Oriented Software Development
FOP	Feature Oriented programming
AOP	Aspect Oriented Programming
AHEAD	Algebraic Hierarchical Equations for
.	Application Design
DSL	Domain Specific Language
VPL	Visual Programming Language

CHAPTER 1

INTRODUCTION

1.1 Background

In most of the product based software development organizations,there is a hierarchical code structure which has several layers included. One layer is built on top of another by using the capabilities provided by the previous or below layer. This hierarchical model helps the developers to use the abstractions and lower level code segments to customize their own code.

Many organizations use this method in order to re-use common software components which later could be customized at higher levels. Hence the lower level coding must be done very carefully, because if there is a mistake in that layer,it will propagate to all the layers which are built on top of it. Coding optimizations and many good practices have to be used when developing lower level codes.

For an example In MillenniumIT Software (Pvt) Ltd ,mainly there are 3 layers. Namely;

- Solution layer
- Product layer
- Technology layer

It could be visually shown as per Fig. 1.1

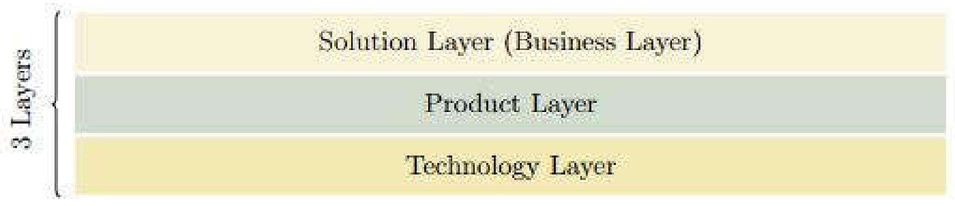


Figure 1.1: Layered product architecture

1.2 Problem Statement

As per the above mentioned layered architecture, the top layers are currently built blindly without analyzing what are the needy code segments and what are the non-needy code segments.

There are so many capabilities that have been provided by product layer, which could be used by the solution layer. But the question arises, "Are those code segments really needed to implement a solution?"

Product layer will provide a complete generalized code, that could be used by any solution layer implementation .What a developer could do is, either comment the unnecessary code segments or not calling the methods that are not required for the particular solution. The solution layer will ignore the unnecessary functionality and implement only the features that are requested by the client.

This is not just related to the codes. The problem exists in middle layer messages too. The message may include some information which are not needed for a particular solution.Also there is a lot of unnecessary components in the business rules and configurations .

There are so many problems that have been raised because of this generalized product components.

This problem also applies to technical documentation. For an example , the product layer documents will have all the functional and non functional information about the product. If some solution layer needs a document which only contains the information about its required features,they will have to manually remove the unnecessary parts of the document to clean it up.

As a summary this will result in following drawbacks in this layered traditional method of software development.

1. Unnecessarily bulky code.
2. Hard to understand by the developers as there are so many unnecessary code segments which are not even used.
3. Increase the size of the messages that are used in the middle-ware which will result in high latency in transmission.
4. Increase the probability of introducing bugs to the existing code.
5. Increase the complexity of code, business rules,configurations and messages.
6. Lengthy technical documentation which contain a lot of unnecessary information

which may even confuse the client or even require additional work to clean the document.

1.3 Objectives

The objective of this research is to find a solution to overcome the above mentioned problems. The solution must cover all the following areas;

- Code.
- Messages.
- Configurations.
- Business rules.
- Documentation.

These could be overcome by a concept called " Feature Oriented Software Development" .The ultimate objective is to provide;

1. A light weighted code that could be easily understood which will only contain the needy features.
2. Light weighted messages that could be easily transmitted which will ultimately help to increase the throughput.
3. Business rules which is only relevant to the particular solution which even could be understood by the clients.
4. High maintainability and clean software components.
5. Capabilities which enhances easy debugging and auditing.

1.4 Feature Oriented Software Development (FOSD)

1.4.1 What is Feature Oriented Software Development?

Feature-oriented software development (FOSD) is a paradigm for the construction,customization, and synthesis of large-scale software systems [1] . A software program is built as a stack of layers. In each layer ,new functionality is added to the bottom or previous layer[2] . Different composition of these layers will create different solutions.

A feature could be described as a unit of functionality in a software program, which will satisfy some particular requirements or any other design goal [1]. After some development, each above mentioned layers could be even considered as a feature. The main idea behind FOSD is to decompose the software program into these little features that could be re-used whenever needed.Ideally from a set of features ,we could develop many different versions of the software and present to the client as per their requirements.

The concept of feature oriented software development has a big history. As per a research carried out by Muller when he worked at Philips research laboratories [22] in 1997,the effort we put to come up with a end software product should be minimal.Therefore FOSD should provide such mechanism to develop a software product in its easiest way consuming minimum effort and cost.

1.4.2 How FOSD could be used to provide a solution

To overcome the issues mentioned in previous sections, we could use this concept of FOSD. Software development could be carried out as a series of feature developments by the product development team. Whereas solution development team can select only the features that have to be delivered to the client.

1.5 Outline

The rest of the report will contain mainly the literature review about the existing and similar solutions which have been already implemented or currently in research. Chapter 2 will give an overview and a detailed summary of the current development in this feature driven software development implementations

Chapter 3 will give a basic insight about the architecture of the proposed solution which is followed by another section that describes the current progress of the research. A detailed time-line as to how this research will be carried out,has been presented in Chapter 3.

In chapter 4, the implementation details of both controller and code generator have been included . Finally in chapter 5 the evaluation of the research has been carried out.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

The very basic idea of Feature Oriented Software Development is to decompose a bigger system in to some smaller features [3]. Then the end system or the product is composed using those basic features.FOSD basically aims at three properties;

- Structure - Features could be structured in order to come up with the desired architecture
- Reuse - Features can be reused throughout the development process
- Variation - With various combinations of features, the designer or the programmer could come up with different variations of the end system

2.2 What is a feature ?

FOSD is not a single method of developing software systems, instead its a combination of many different methodologies , languages,tools,theories,ideas etc.[3]. A feature connects all the above mentioned different elements.There are several definitions of a feature.

1. Kang et al. - A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [5]
2. Kang et al. - A distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained [4]
3. Bosh - A logical unit of behavior specified by a set of functional and non-functional requirements [7]
4. Classen et al. - A triplet, f = (R,W, S), where R represents the requirements the feature satisfies, W represents the assumptions that the feature takes about its environment and S represents its specification [6]

The feature definitions could be even classified in the perspective of Problem Space and Solution Space [8] . As per Fig. 2.1 , the problem space describes the basic requirements of a software system whereas the solution space describes how those problems have been satisfied or solved and how the requirements have been fulfilled & implemented.

The first three definitions of a feature, which are described above, could be classified in the problem space whereas the last definition could be classified in the solution space.

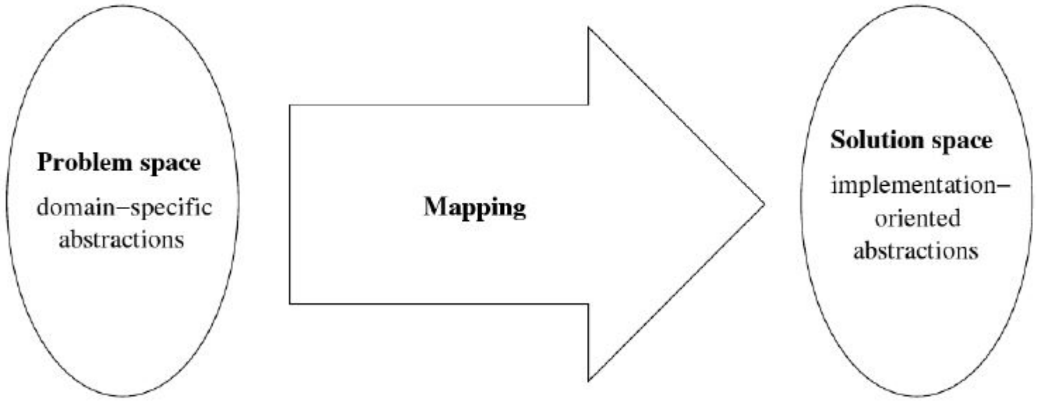

Problem space
domain–specific abstractions
Mapping
Solution space
implementation–oriented abstractions

Figure 2.1: Problem space and solution space [8]

The mapping should be done carefully and in an efficient manner which maintains the software or documentation re-usability.

2.3 Different phases of feature oriented software development

As per the journal paper [3] , there are 4 main phases in FOSD, namely

1. Domain analysis

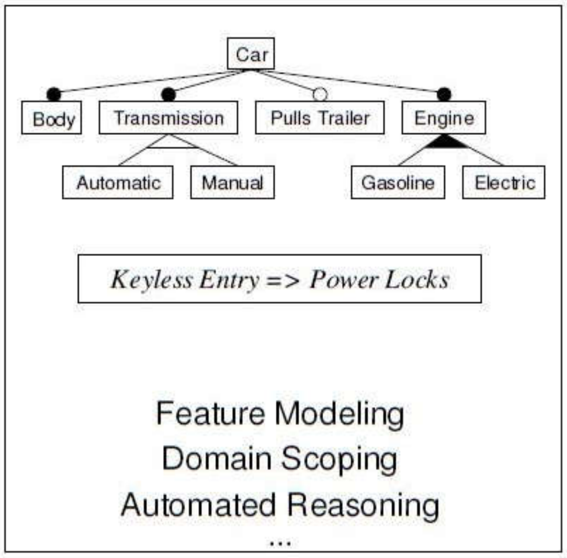


Figure 2.2: Domain analysis [3]

2. Domain design and specification

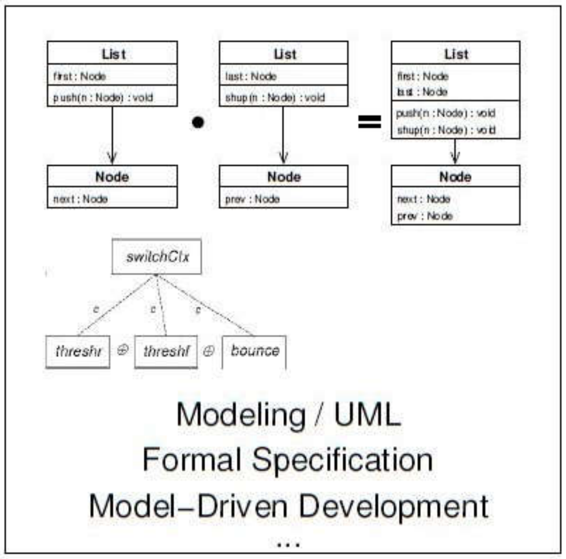
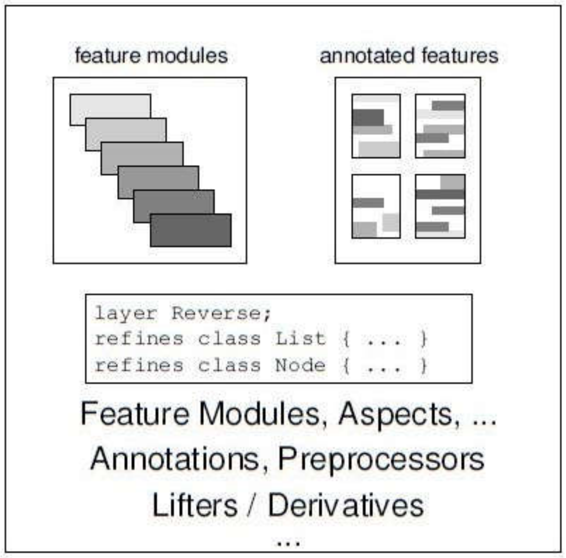


Figure 2.3: Domain design and specification [3]

3. Domain implementation



```
layer Reverse;
refines class List { ... }
refines class Node { ... }
```

Figure 2.4: Domain implementation [3]

4. Product configuration and generation

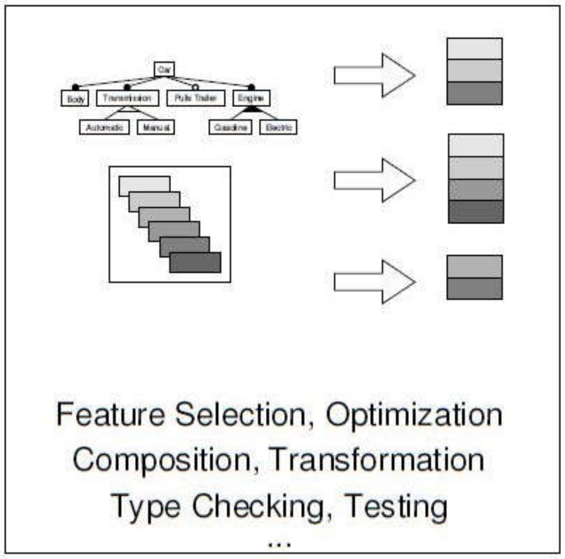


Figure 2.5: Product configuration and generation [3]

In each step, the features that are needed to build an end software product or a software product line ,are identified.The supportive features and non supportive features are identified correctly.

More importantly there are 3 main lines of research in FOSD,namely;

1. Feature Modeling
2. Feature Interaction
3. Feature Implementation

2.3.1 Feature Modeling

From the research that has been carried out by Kang et al [5] ,they have structured the problem space using the concept of features. A standard example is shown in Fig. 2.6 by using feature diagram notations [8]

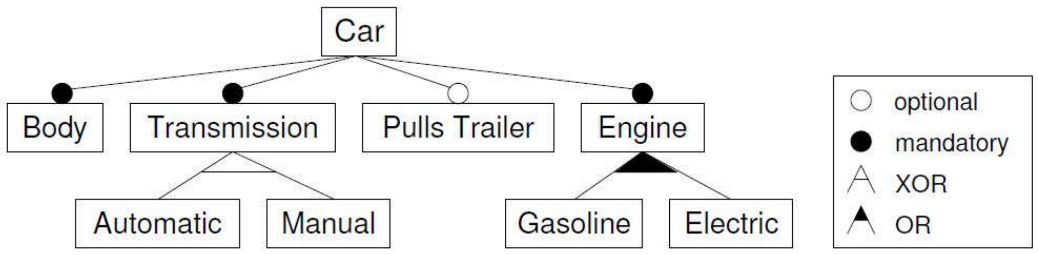


Figure 2.6: Feature Modeling [3]

2.3.2 Feature Interaction

The concept of feature interactions arise from the field of telecommunication.The software based researches have used the same concept in order to develop feature oriented software development methodologies.

Feature Interaction is where two features show different unexpected results or outcomes when they are interconnected, compared to the behavior that could be observed when the two features function independently.The basic example they have taken is, call forwarding and call waiting functionality. We can clearly predict the behavior when the two functionality is run independently. But we cannot exactly predict what could happen if the two functions run interdependently. [19]

Therefore we must keep in mind that feature interaction is a major problem in feature oriented software development.

2.3.3 Feature Implementation

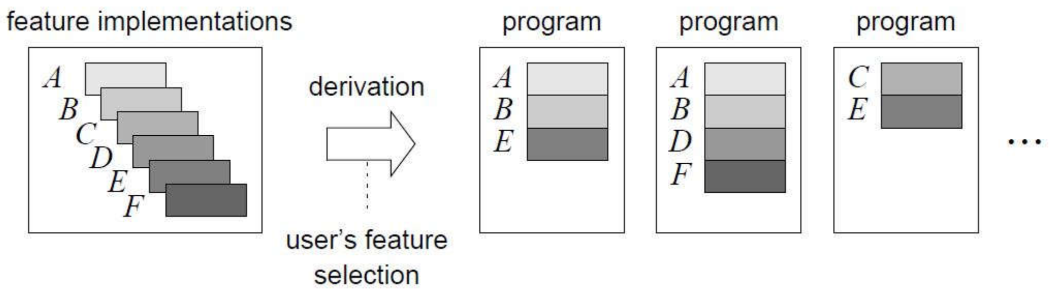


Figure 2.7: Feature Implementation [3]

Finally comes the feature implementation, which separates the base code from its additional features as per Fig. 2.7.Prehofer was the first one to introduce the concept of features in software context [3].

These above main lines of researches converge to build an end feature oriented software development methodology as per Fig. 2.8

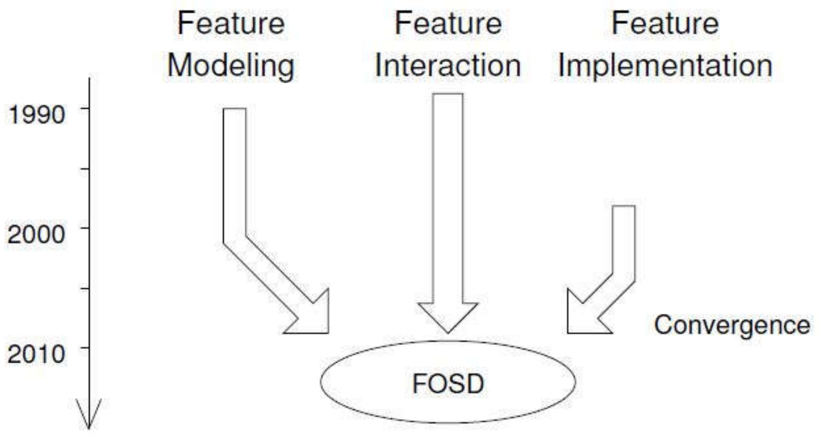


Figure 2.8: Convergence of each step in FOSD [3]

2.4 Related Implementations

2.4.1 FeatureC++

FeatureC++ is an extension to C++ which supports feature oriented programming (FOP) [9] and aspect oriented programming (AOP). FeatureC++ has many features like multiple inheritance etc.Here AOP is a similar concept same as FOP which tried to solve problems such as increased complexity of software programs,reduced understandability and customizability [13] [14]

FeatureC++ has been implemented using a concept called Mixin Layers.Mixins implements the class fragments as per Fig. 2.4. It shows a stack of three mixing layers [9]. Similarly they are talking about child and parent mixing layers too.

2.4.2 Feature Implementation

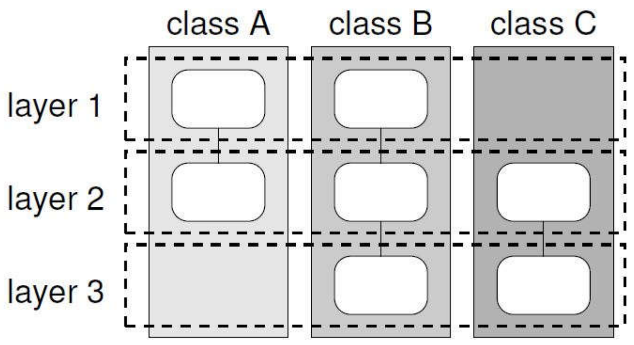


Figure 2.9: Stack of Mixin Layers [9]

In order to develop FeatureC++ they have used Jak [12] syntaxes which are accepted by several parties.They have mainly focused on resolving the problem which are mentioned in Table 2.1

Problem	Solution
Homogeneous crosscuts	Pointcuts and advices
Interface extensions	Method interception, argument passing by aspects
Hierarchy conformity	Refine only structure relevant Mixins
Dynamic crosscutting	Use specific pointcuts (cflow , etc.)
Method extensions	Wildcards in pointcut expressions

Table 2.1: Problems and Solutions given by FeatureC++ [9]

Even though FeatureC++ has some insights about the problem, it does not directly solve our main problem and does not indicate how FOSD could be used to come up with the required solution.

2.4.3 Algebraic Hierarchical Equations for Application Design (AHEAD)

In AHEAD, they have used a concept called "Step Wise Refinement" [12] . Step Wise Refinement is used to develop a complex system from a simple system by adding required features incrementally. AHEAD shows that a software can have a hierarchical mathematical structure which could be expressed as a nested set of equations.

Mainly this paper describes how step wise refinement scales to the simultaneous synthesis of multiple programs and multiple non-code representations written in different languages

They start their research using GenVoca model [15] which describes how an individual program could be represented using an equation. Then they move on to Algebraic Hierarchical Equations for Application Design model that generalizes the equational specification of multiple software programs.

2.4.4 Model Concepts

- The initial program is just constants and they have used refinements to add features to the initial program to come up with the desired end program.

f // program with feature f

g // program with feature g

Then refinements take the program as an input and then make a feature augmented program as the output

i(x) // adds feature i to program x

j(x) // adds feature j to program x

Then a multi-feature program can be expressed as per below equations.

app1 = i(f) // app1 has features i and f

app2 = j(g) // app2 has features j and g

app3 = i(j(f)) // app3 has features i, j, f

The equations show both the implementations and the features.Apart from that they have used following 4 ideas to generalize the GenVoca model.

1. An application has other representations beyond source code
2. A module is a containment hierarchy of artifacts that can include multiple representations of an application
3. A scalable notion of refinement should be able to refine all representations in a consistent fashion
4. Principle of Uniformity

Figure 2.5 illustrates how AHEAD composer tool works.

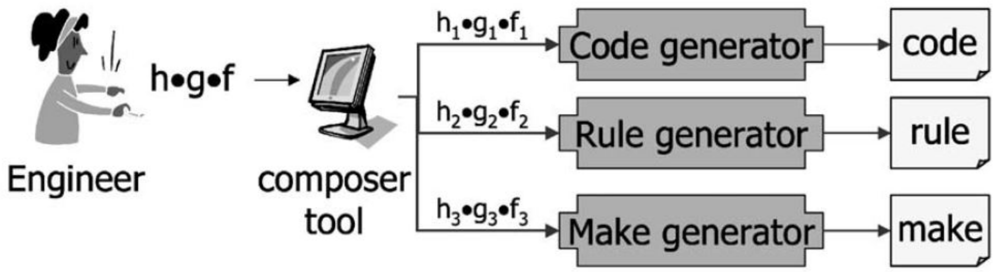


Figure 2.10: Organization of AHEAD generators. [12]

2.5 Access Control in Feature Oriented Programming

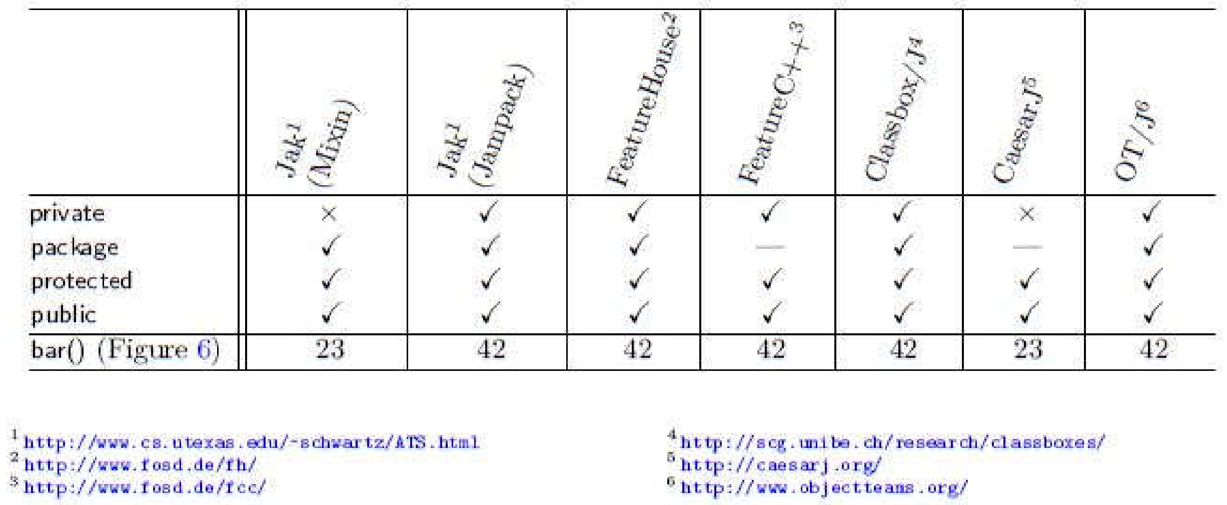
In feature oriented software development process, a program could be decomposed in to a set of features. Those features could be developed in an isolated manner or independently.A feature module encapsulates exactly the code that contributes to the implementation of a feature[10] [16]. Access control plays a major role in these modularity concept in order to provide required visibility to expose or hide internal details of each module or feature. In other words this is a function which exhibits the encapsulation of each feature or module.If those encapsulations are not done properly the end program could mis-behave unexpectedly.

Many feature oriented languages such as FeatureC++ [9] ,AHEAD/Jak [12] and FeatureHouse [17] aim at feature modularity. Those feature modules have to be implemented in such a way that their internal implementation details are hidden and the functionality is provided via interfaces[18]. However it is really hard to implement such features which exhibits correct modularity. Those feature interactions and access controlling is described in this research paper.

As per Table 2.2 we can compare different feature oriented languages with respect to their rules in accessing filed from refinement and program behavior [10] . Therefore it is clear that when we design or come up with a feature oriented language ,we need to pay our attention on well defined semantics. There must be access modifies that are more specific to the feature oriented software development methodologies.

In the above mentioned research paper they have focused on some access modifiers that helps feature oriented programming,namely [10]

- Modifier feature
- Modifier subsequent
- Modifier program



	Jak[1] (Mixin)	Jak[1] (Jampack)	FeatureHouse[2]	FeatureC++[3]	Classbox/J[4]	Caesar.J[5]	OT/J[6]
private	×	✓	✓	✓	✓	×	✓
package	✓	✓	✓	—	✓	—	✓
protected	✓	✓	✓	✓	✓	✓	✓
public	✓	✓	✓	✓	✓	✓	✓
bar() (Figure 6)	23	42	42	42	42	23	42

[1] http://www.cs.utexas.edu/~schwartz/ATS.html
[2] http://www.fosd.de/fh/
[3] http://www.fosd.de/fcc/
[4] http://scg.unibe.ch/research/classboxes/
[5] http://caesarj.org/
[6] http://www.objectteams.org/

Table 2.2: Which members of class could be accessed by a refinement in each language[10]

2.6 Domain Specific Languages (DSL)

Domain specific languages are implemented in order to hide low level language details and present it in a abstract way to its users.There are domain specialists and there are also language specialists. But there is very few who is good at both programming language and the domain. The domain specific languages helps to fill this technical gap and provide very advance solutions.

The domain specific language goes in line with feature oriented software development as per the research paper based on the third international workshop on feature oriented software development [20] . It could be developed to a database system, banking system or even for a text processing system.It is mentioned that conference was held in order to enhance their collaboration in doing researches in feature oriented software development.

But in contrast some argue how to develop domain independent feature oriented program .Mehran and his team has developed a concept called ciFeature which is an abbreviation for context independent features [23].They have developed something called a feature factory and have described how those could be used to develop context independent features.

Feature oriented software development methodologies could be even used to analyze a particular domain or a context [24]. As per Herbert and his team,FOSD could be used to analyze their energy aware self adoptive software.They use a concept called energy aware feature modeling in order to develop their research.This has also been described in section 2.3.1 too. They claim that a feature can be made of several sub features as per the equation 2.1 [26]

$$f_i = \{f_{i \cdot 1}, f_{i \cdot 2}, f_{i \cdot 3}, ... f_{i \cdot m}\} \tag{2.1}$$

where i denotes any feature and m denotes a finite number related to its sub features.

As per Fig. 2.11 each feature could be analyzed in a particular domain using its sub features.It could be even analyzed as a divide and conquer method as small features will eventually develop the entire system as per equation 2.2.

$$eg.f_{Location} = \{CoarseGrained, FineGrained\} \tag{2.2}$$

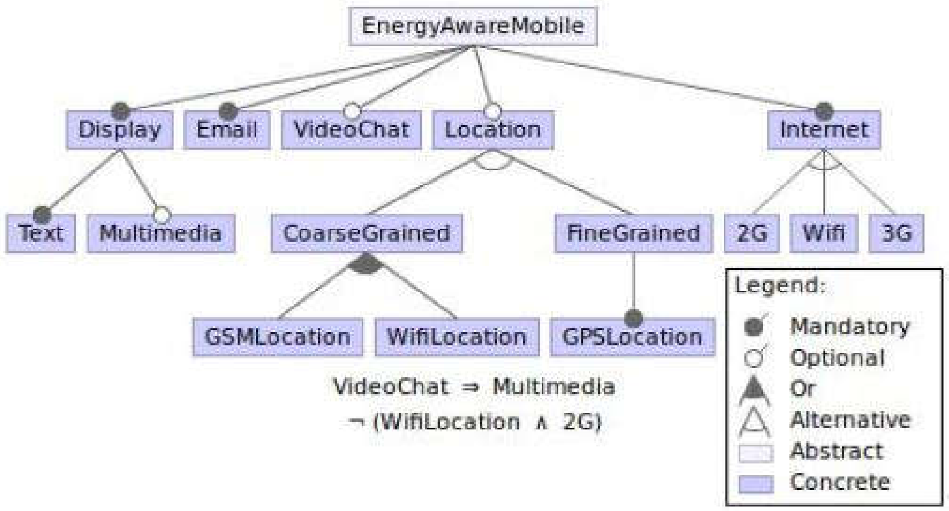


Figure 2.11: Energy Aware Feature Model [26]

2.7 Summary of the Literature Review

The literature review mainly focuses on feature oriented software development concepts and what are the existing solution or related work. By referring to the research papers it can be seen that there is no proper solution to our initial problems which are mentioned in the problem statement (section 1.2).

Therefore the main motivation to continue the research is to find the desired solutions to the mentioned problems in a more realistic way.The existing researches cannot be directly applicable to resolve them.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 High Level Architecture

The ultimate objective of this research would be to develop some kind of a code or text generator which will collect and correlate different features to build a complete end product. In order to implement this, the product layer will have to be decomposed into several features. It will be a challenging task and sometimes some features may not be able to decompose due to their complexity. In such scenarios we will have to develop some complex features which will provide a set of features instead of one feature.

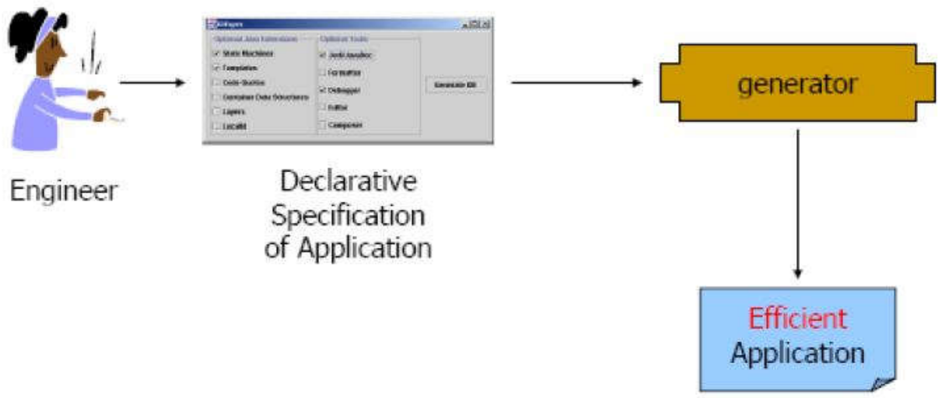


Figure 3.1: Code generation based on selected features [3]

Solution layers of the organization will use each feature provided by the product layer in order to come with the desired solution. As per Fig. 3.1, a code generator will be the final outcome of this research, which will let the user to generate the final efficient code based in the selected features. As per Fig. 2.7 the developers in the upper layers could select the required features from a pool and then generate the code as per the requirement.

For the software development organizations there will be some added advantages because of this feature oriented software development methodology. Each feature could be developed using each scrum team in accordance with the agile methodologies.The team will have to take responsibility of each feature and there will be an architect who will manage the interoperability and interaction of features.

Once the code generator generates the code, those could be injected to a core controller where the features could present their behavior. The controller will be implemented in such a way that ,it could load the instruments and other data first and facilitate the features to function properly.

As per the requirement of a stock exchange , the core controller should be implemented using a language which could run very fast . For that purpose C++ language is chosen to develop the core controller.

The code generator will be implemented using the Java language in order to facilitate the features in Antlr and StringTemplates parser .

3.2 Progress

Currently the detailed literature review has been carried out and a background search about the existing implementations have been analyzed. After those preliminary studies, a detailed design has been drafted before implementing the final products, core controller and the code generator. First the code generator was implemented and then the core controller is developed in order to evaluate the auto generated codes.

Apart from main controller and the code generator, a Java based automated testing tool has been developed in order to validate the results.The research has been fully implemented under the selected criteria and limitations. The study limitations has been mentioned in the chapter 6 under study limitations.

3.3 Evaluation Methodology

For the final evaluation for this research, I have planned to provide a code generator or a text generator which helps to develop feature oriented software programs. Each feature will have to implement independently .When the required features are selected, the final product will be generated automatically.

Apart from that, I have planned to demonstrate how the code generator works . A user manual will be presented along with full documentation.

In order to evaluate the code generator the core controller could be used. The initial design itself helps to evaluate the system and it is a real advantage of this design. The auto generated code will provide the inputs to the core controller so that it can exhibit the behavior of each feature. Several combinations of the features could be evaluated and results could be compared either with manually calculated results or even against a controller which has been implemented without using feature oriented designs.For that, an automated testing tool has been designed and developed so that the testing process becomes more professional and accurate. The expected results will be compared against the actual results. If the model is correct and accurate, both expected results and the actual results should be the same.

A time line as to how this research has been carried out is shown in Fig. 3.2 in section 3.4. The research time-line was maintained smoothly through the period by getting feedback from different parties in order to come up with the optimum solution.

3.4 Time line

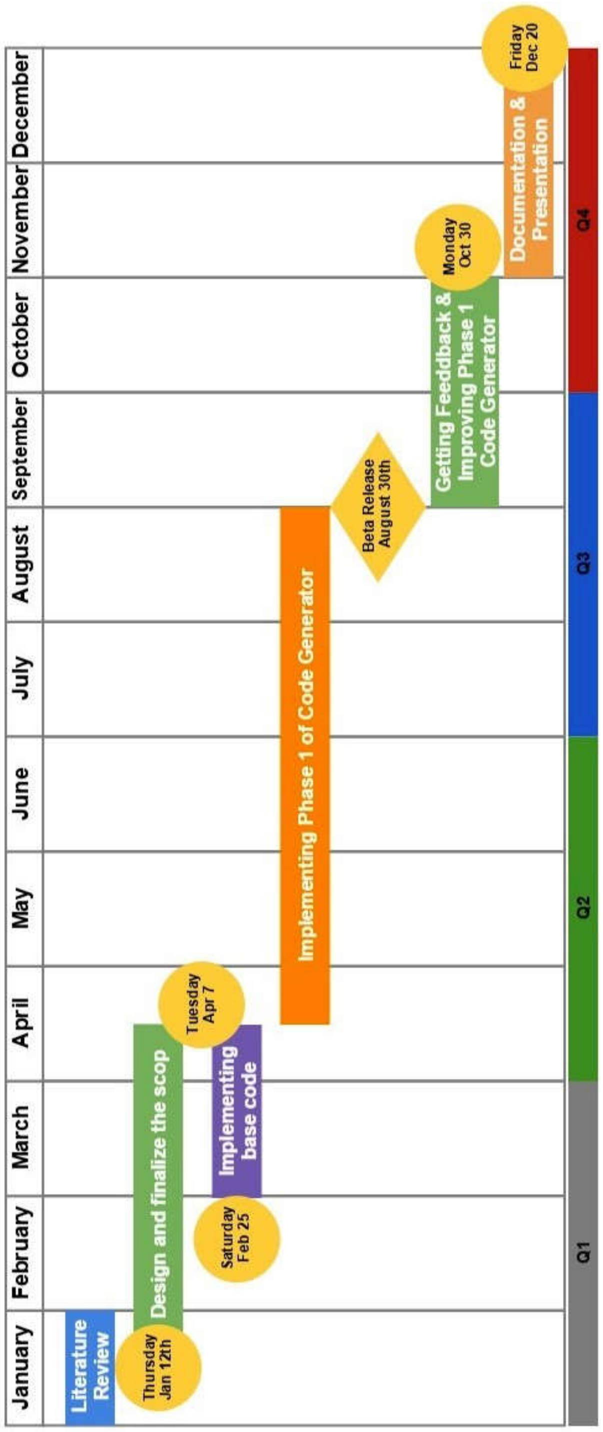


Figure 3.2: Research Time-line

CHAPTER 4

SYSTEM ARCHITECTURE AND IMPLEMENTATION

4.1 System Overview

The system can be mainly divided in to two components. Namely;

- Code Generator
- Controller

Code generator generates the main feature classes using the defined grammar and the templates. Controller is the one which facilitates the functions of each feature .Therefore the implementations are discussed in separate sections in order to have a better understanding.

4.2 System Architecture

The over-roll architecture could be described with respect to the code generator and the controller.

4.2.1 Architecture of Code Generator & Controller

The main objective of this research is to develop a feature oriented and domain specific methodology to build a stock exchange system. In order to implement that goal we have to come with a domain specific language (DSL) which will help to include business logic and components in to the system.

As per figure 4.1 the main input for the system is the DSL. The DSL input is given to a parser. For this purpose we could use any parser but we have selected Antlr version 4.7 (Antlr4) as the parsing tool in order to implement this research.The code generation process has been explained by P. W. Burke in his research on automatic code generation using model driven design[43] . The automatic code generation could be considered as an art. Its really fascinating when the final code is generated as per the given DSL.The implementation of the DSL has been described in section 4.3.1 in detail.

Antlr parses the DSL and information is retrieved . Once the information is retrieved , the Java code will convert it to the required output language using String Templates. The translation is done smoothly with adhering to the final requirement of being feature oriented.

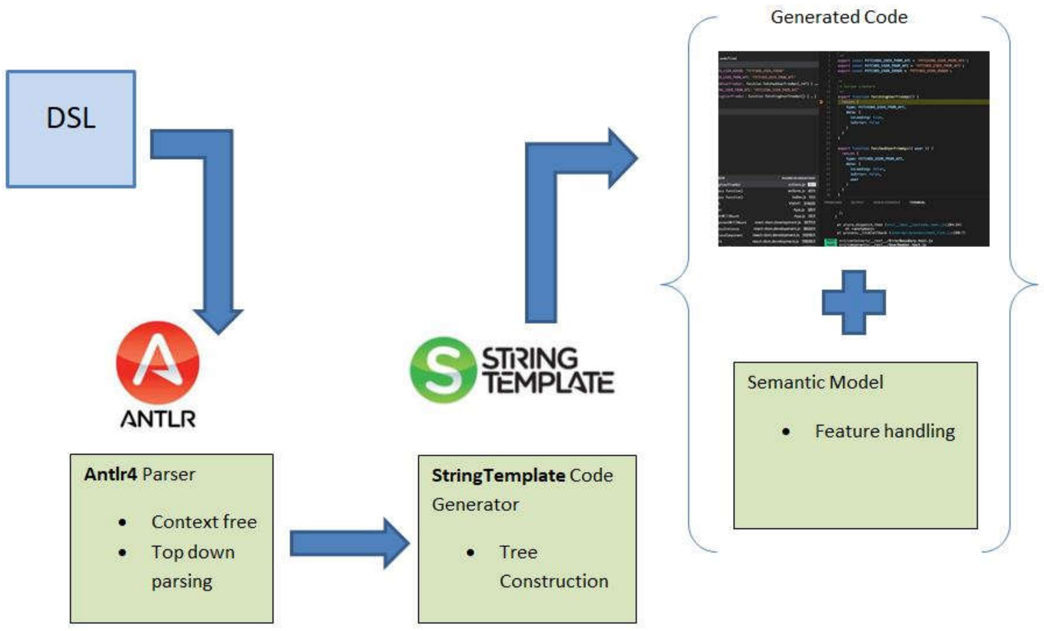


Figure 4.1: Architecture of Code Generator

Now that we have our generated code we can use them along with the semantic model in order to make the Controller. The controller will have its semantic model in order to provide its core functionality whereas generated codes provide the features that could be used by the core.

There are specific properties that should be there in any feature. The feature must be described in such a way that it could be solely implemented independent of the other features. For an example if we declare a variable ,we have to include it in all features which requires that field. In that case we can take the union of the fields where we could come up with the class structure which has all the required fields.

The self explanatory Fig. 4.1 shows how DSL is parsed using Antlr parser and how StringTemplates would help to generate the output codes, also how the generated codes are combined with the semantic model to implement the controller.

Implementations of each component will be described in section 4.3 onward.

4.2.2 Class Diagram of Code Generator

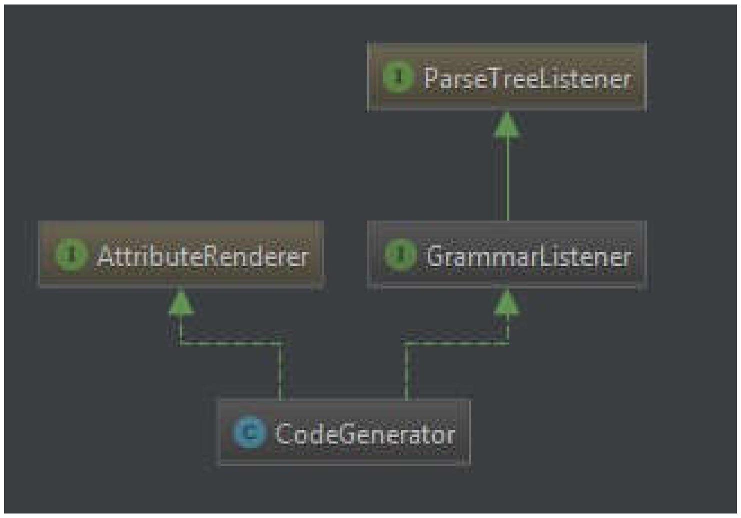


Figure 4.2: Code Generator

Code generator is the main achievement in this research. It will help to generate feature oriented software components when the input is injected through a domain specific language.

As per Fig. 4.2 code generator implements AtributeRenderer and a listener called GrammarListner. GrammarListner that has been inherited from ParseTreeListner, is the listener which has all the callback functions of Antlr4 grammar. Code generator will receive all the callbacks as an when the grammar is parsed through and we can use those callbacks in order to generate the required output.

Also Code Generator has implemented AttributeRenderer class which is used to manipulate and change the formats at code generation run time. For an example,the cases of each word can be manipulated using this attribute renderer.The word can be either converted to upper case,lower case or even camel case or into pascal case depending on the application .Another example is, the prefixes of each attribute have been changed at code generation in order to provide a code which adheres to the C++ coding standards and practices.

The renderer is used at the run time of code generation with StringTemplates. It will render the output using the renderer before generating the final output.

4.2.3 Class Diagram of Controller

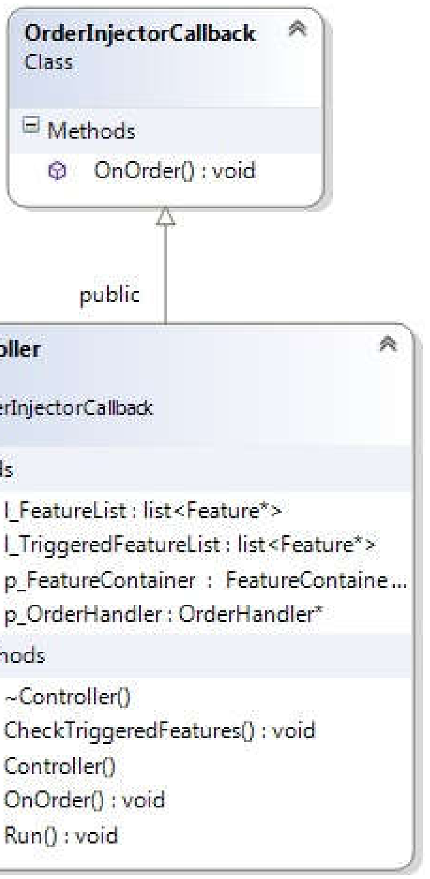

OrderInjectorCallback
Class
Methods
OnOrder() : void
public
Controller
Class
→ OrderInjectorCallback
Fields
l_FeatureList : list<Feature*>
l_TriggeredFeatureList : list<Feature*>
p_FeatureContainer : FeatureContaine...
p_OrderHandler : OrderHandler*
Methods
~Controller()
CheckTriggeredFeatures() : void
Controller()
OnOrder() : void
Run() : void

Figure 4.3: Controller

The controller consists of the semantic model and the generated codes. The generated code is directly taken from the code generator .Controller has a callback from OrderInjector which will help to receive incoming orders to the system.

It has a list of features and also a list of triggered features. The list of features will contain all the features that have been generated by the code generated in the previous stage. Apart form that, it also has a triggered feature list in order to maintain the list of features that are applicable to the current order injected by the OrderInjector.

It is strongly advised not to change any of the generated codes when implementing any feature throughout the life cycle of the exchange product. The reason for that is when you change the auto generated code segments the users will have to manually merge the components when generating the code again.

The complete implementation of this controller is described in section 4.3.3 onward.

4.3 System Implementation

4.3.1 Implementation of Code Generator

Implementation of the code generator is quite interesting as it has some unique features. It is not just converting some piece of code to another existing language. It has several elements which makes it more advanced in several ways.

The starting point of the code generator is the introduction of a domain specific language (DSL). The domain that is addressed throughout this research is the stock exchanges domain.

This implementation can also be considered as a code refactoring ,which means non feature oriented software product could be converted to a feature oriented software product throughout this research. Similar concepts have been discussed by Roberto and his team from which I have taken references [28]

4.3.2 Introduction of Domain Specific Language (DSL)

When the focus is narrowed down to a specific domain , it will be easier to develop a language that is tailor made to that domain .It will also be more effective when it comes to use the language by the users because of its domain specific nature. The language will contain some key words which are related to that domain.

Code generation would be much efficient and effective when the key words are defined in the correct way. Also it is advised not to use keywords everywhere , which will confuse the end users.

There is an interesting research carried out to investigate when and how to develop domain specific languages by Mernik, Marjan, and his team [39]. They claim that there is a massive gain and advantages in developing a domain specific language compared to using a general purpose language to a particular industry.

The business rules will be presented in the form of DSL. These will be later converted to actual code in order to use it in our semantic model.

4.3.3 Use of Antlr to define the grammar

When a domain specific language is developed , we may have to use a parser to parse through the newly defined keywords and rule. Parsing a word set is not the main part of this research hence I have used a parser called Antlr. Antlr has some advanced features that will help us to parse through the rule or the DSL. In the research Antlr 4.7 is used which is the latest version.

There is a lot of research done using this Antlr tool.A domain specific language has been developed using Antlr in order to process transactions by Neeraj and his team [37] .They have developed a language called XBRL in order to program transaction processing system even without having much knowledge about low level programming.This research paper has shown how a domain specific language should develop and each step is described in a very descriptive way.The way a domain specific language should be developed is also mentioned by Van Deursen and his team [38]. Another application of Antlr parser is described in a research by Danyang Cao and Donghui Bai which explains how a SQL parser is developed [40].Another application is, use of Antlr to transform C language to a high level language which has been implemented by Yueming Zhao and his team [41]. These researches describe the use of parsers and how to parse the domain specific language in order to extract the required information.

Another advantage in using Antlr as the parser, we could directly get the class object as the parsed output. In that case the listener output will be a Java object not just a string output. In the implementation I have tried to push more things to the Antlr grammar and to the templates by keeping the Java code generator clean.

In Antlr, we define the keywords that were mentioned in the above section. Use of keywords will be described in section 4.3.3 which also shows a sample code written using the newly introduced DSL.

Mainly there are several types of data structures or reference data containers used in the DSL and also we will have to define some actions that has to be performed throughout the execution of the controller.Below sub sections will describe how the reference data is loaded and how the behavior is defined.

Defining required reference data.

Namely there are two major types of entities that we have to define in our domain specific language.

1. **Base type entities.**

   Base type entities will not be loaded by the reference data loader. The user will have to provide the input when he uses the entity in the semantic model. Base type entities could be loaded either using an external xml file or using some other output generated at the controller. The keyword for a base type entity is defined as per the below example of an Order entity.

   ```
   eg:
   Entity  Order  (Type = Base)
   {
       // Fields
   }
   ```

2. **Persistent type entities.**

   In contrast ,persistent type entities are loaded directly from the reference data loader. The inputs are loaded from an xml files using c++ boost library features. The keyword for a persistent type entity is defined as per the below example of an Instrument entity.

   ```
   eg:
   Entity  Instrument  (Type =  Persistent )
   {
       // Fields
   }
   ```

In the same way there are three types of fields used.

1. **Key fields.** When we define an entity we need to have a key in order to identify the entity uniquely. In order to have that ability we use a key field for each of the entities. A key field helps the entity to get loaded to the reference data containers and also when the entity information is retrieved. A key field is defines as per the below example.

   ```
   eg:
    String  instanceID  (Key Field);
   ```

2. **Base fields.** Base fields are directly loaded from the external xml files same as base entities and they are converted to both object type and value type in the generated outputs.As they are the most common field type ,we are not defining the field as base type field explicitly. They are defined as per below

   ```
   eg:
    Integer  side;
   ```

3. **Stat fields.** Some fields are not loaded from the exiting entities. Therefore these have to be handled separately and they are not loaded from the reference data containers. They are defined as stat fields as per below

   ```
   eg:
   OrderBook orderBook (Stat  Field);
   ```

Defining behavior.

The reference data is there and now we need to define the behavior or actions. Actions are defined inside an event. There are pre-defined events as per below which are specific to the stock exchange domain.

- Init
- PreMatch

- Match
- PostMatch
- Clear

These predefined events could either be loaded from an xml file or just hard code as per the initial implementation.Inside an event we can define user actions. There can be either validations or any other action. There are some events like OnOrder and AddToOrderBook etc. The users could use the keywords in their DSL code in order to generate the required feature.

An event is defined as per below format.

```
Event event_name
{
    // actions  to  be  performed
}
```

Apart from these, there can be feature specific attributes. These can also be defined as per the example shown in section 4.3.4. The example shows how the limit order functionality has been implemented as a feature using our domain specific language.

4.3.4 Sample DSL Code

```
Feature LimitOrder // Feature name
{
    Define Integer orderTypeLimit = 1;

    Entity Order (Type = Base)    // Defining a "Base" type entity
    {
        String instrumentSymbol;
        String orderID (Key Field);   // Defining the key field of the entity
        Integer side;
        Integer size;
        Integer orderType;
        Integer tif;
        Instrument instrument;
        Integer action;
        Float price;
        Integer orderStatus (Stat Field); // Defining fields that have be set by the user
    }
    Entity TradingParameter (Type = Persistent ) // Defining a " Persistent " type entity
    {
        String instanceID (Key Field);
        Boolean enableLimitOrders;
    }
    Entity Instrument (Type = Persistent )
    {
         String symbol;
         TradingParameter tradingParameter;
         Integer instrumentIndex (Key Field);
         String lastTradingDate;
         OrderBook orderBook (Stat Field);
    }
    Event Init
    {
        Validate order.orderType equals to orderTypeLimit,5001/ Invalid Order Type hence feature not triggered;
    }
    Event OrderSubmit
    {
        Validate order.instrument.tradingParameter.enableLimitOrders equals to true,2031/Limit Orders Trading Parameter is not enabled;
    }
    Event Match // Defining an event
    {
        OnOrder(order);          // Feature actions
    }
    Event PostMatch
    {
        AddToOrderBook(order);
    }
    Order order; // Feature attributes
}
```

Using the DSL , following code segments will be automatically generated which are shown in section 4.3.5 and 4.3.6 Only the main auto generated code segments are included in the report.

4.3.5 Auto generated C++ code

```
#include "LimitOrderFeature.h"
#include "Logger.h"

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
LimitOrderFeature :: LimitOrderFeature ()
{
    p_OrderHandler = nullptr ;
    p_Order = nullptr ;
}

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
LimitOrderFeature ::~ LimitOrderFeature ()
{
}

// Get Methods
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
Order∗ LimitOrderFeature :: GetOrder_p()
{
   return p_Order;
}

// Set Methods

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
void LimitOrderFeature :: SetOrder(Order∗ order)
{
   p_Order = order;
}

// Set Context
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
void LimitOrderFeature :: SetContext(Order∗ order )
{
    SetOrder(order);
}

// Set Order Handler
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
void LimitOrderFeature :: SetOrderHandler(OrderHandler∗ pOrderHandler)
{
   p_OrderHandler = pOrderHandler;
}

// Events
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
bool LimitOrderFeature :: OnInit ()
{
   CLogger::GetLogger()−>Log("LimitOrderFeature firing Event:OnInit  for Order ID: [%s]",p_Order−>GetOrderID().c_str());

   if (!( GetOrder_p()−>GetOrderType() == orderTypeLimit ))
   {
        CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnInit Returning  false  due to  :  / Invalid  Order Type hence  feature  not  triggered ;  Order ID:
                [%s]  Reject  Code: [%d]",p_Order−>GetOrderID().c_str(),5001) ;
        p_Order−>SetOrderStatus(ORDER_STATUS_REJECTED);
        return   false ;
    }

    CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnInit − Returning  true  for  Order ID :  [%s]",p_Order−>GetOrderID().c_str()) ;
    return   true ;
}
// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
bool  LimitOrderFeature :: OnOrderSubmit()
{
    CLogger::GetLogger()−>Log("LimitOrderFeature firing  Event:OnOrderSubmit  for  Order ID:  [%s]",p_Order−>GetOrderID().c_str()) ;

    if (!( GetOrder_p()−>GetInstrument_p()−>GetTradingParameter_p()−>GetEnableLimitOrders() == true ))
    {
        CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnOrderSubmit Returning false  due  to  :  /Limit  Orders  Trading  Parameter  is  not  enabled;  Order
                ID:  [%s]  Reject  Code: [%d]",p_Order−>GetOrderID().c_str(),2031) ;
        p_Order−>SetOrderStatus(ORDER_STATUS_REJECTED);
        return   false ;
    }

    CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnOrderSubmit − Returning true for  Order ID :  [%s]",p_Order−>GetOrderID().c_str()) ;
    return   true ;
}
// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
bool  LimitOrderFeature :: OnPreMatch()
{
    CLogger::GetLogger()−>Log("LimitOrderFeature firing  Event:OnPreMatch  for  Order ID:  [%s]",p_Order−>GetOrderID().c_str()) ;

    CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnPreMatch − Returning true for  Order ID :  [%s]",p_Order−>GetOrderID().c_str()) ;
    return   true ;
}
// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
bool  LimitOrderFeature :: OnMatch()
{
    CLogger::GetLogger()−>Log("LimitOrderFeature firing  Event:OnMatch  for  Order ID:  [%s]",p_Order−>GetOrderID().c_str()) ;
    p_OrderHandler−>OnOrder(p_Order);

    CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnMatch − Returning true for  Order ID :  [%s]",p_Order−>GetOrderID().c_str()) ;
    return   true ;
}
// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
//
bool  LimitOrderFeature :: OnPostMatch()
{
    CLogger::GetLogger()−>Log("LimitOrderFeature firing  Event:OnPostMatch  for  Order ID:  [%s]",p_Order−>GetOrderID().c_str()) ;
    p_OrderHandler−>AddToOrderBook(p_Order);

    CLogger::GetLogger()−>Log("LimitOrderFeature Event:OnPostMatch − Returning true for  Order ID :  [%s]",p_Order−>GetOrderID().c_str()) ;
    return   true ;
}
```

4.3.6 Auto generated C++ header

```
#pragma once

#include "Feature.h"
#include "Order.h"

#define orderTypeLimit 1

class LimitOrderFeature : public Feature
{
  public:

      // Constructor
      LimitOrderFeature();

      // Destructor
       ~LimitOrderFeature();

      // Get Methods
       Order* GetOrder_p();

      // Set Methods
     void SetOrder(Order* order);

       // Set Order Handler
       void SetOrderHandler(OrderHandler* pOrderHandler);

       //Events
       bool OnInit() override;
       bool OnOrderSubmit()override;
       bool OnPreMatch()override;
       bool OnMatch()override;
       bool OnPostMatch()override;

       // Set Context
       void SetContext(Order* order );

  private:

      // Attributes
       Order* p_Order;

       OrderHandler* p_OrderHandler;
};
```

4.3.7 Use of StringTemplates to generate the code

Now that we parsed the DSL using Antlr, we have to convert the code into required output language form. In our case, we will need to convert the code to c++ as our Controller is written in c++ language. We can convert to any language by just changing the template that we use. That is a major advantage in using this kind of a design. It could be either converted to c++, Java or even Python depending on the application.

We have used StringTemplates as our template tool.

4.3.8 Challenges faced

One of the challenges I had to face is to implement the conditional statement inside a validation. The conditional statement will need to be converted in to a c++ code in the code generation time. The conditional statement would be a complex one,but we need to convert it somehow to our desired output type. This was bit challenging.

Also the reference data loading was very challenging as to how each type of reference data is automatically loaded without very less user interaction.

Initially I had to analyze how a controller should work and also acquire the domain knowledge on stock exchanges. That was also a challenge for me and I had to develop a separate application in Java in order to understand the behavior of a matching engine in a stock exchange system.

4.3.9 Key assumptions made

There are some key assumptions that we are relying on throughout this research. These assumptions can be removed one by one by doing further research in the future.

One such key assumption is that, we assume that there will be no interaction of features with each other. We are not addressing the cross behavior of features in this research. When features are integrated to the semantic model, each feature is independent and does not reply on any other feature. Each feature can be run without depending on any of the other feature event,action or validation.

4.3.10 Implementation of Controller

Implementation of the controller is described by using each class diagram as per below. The role of each class is defined separately in order to present how it has been implemented.Mainly the systems of MillenniumIT was studied in order to implement this controller . Apart from that Swiss stock exchange system [35] and Nepol stock exchange system [36] were studied in order to have a better understanding and to gather necessary requirements in developing this domain specific language.

First an entity class is defined as per Fig. 4.4 .An entity can have multiple instances. Each instance will have a unique identity and each instance will be stored in a list and also in a map . There are two maps used in order to find instances by their index or name (Unique keys).

List of entities is used to send as an input to the StringTemplate for the code generation.An entity is used to store information of an order or an instrument etc.

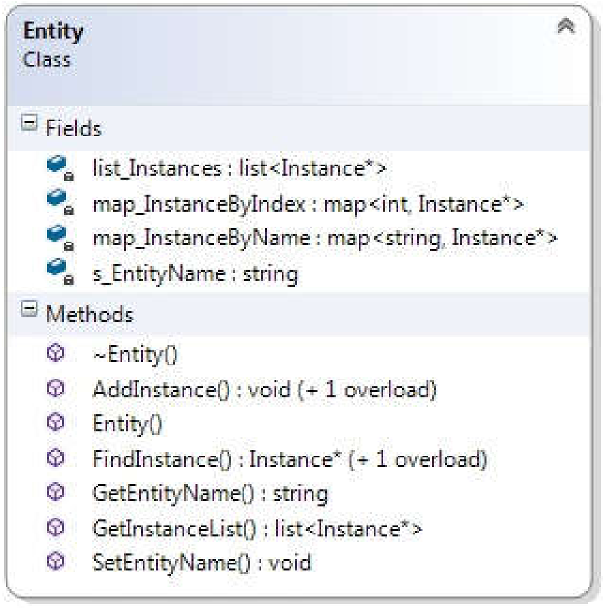

Entity
Class
Fields
list_Instances : list<Instance*>
map_InstanceByIndex : map<int, Instance*>
map_InstanceByName : map<string, Instance*>
s_EntityName : string
Methods
~Entity()
AddInstance() : void (+ 1 overload)
Entity()
FindInstance() : Instance* (+ 1 overload)
GetEntityName() : string
GetInstanceList() : list<Instance*>
SetEntityName() : void

Figure 4.4: Entity

Next the feature implementation will be taken into consideration which is one of the main components in this design. A feature is something which can provide some functionality to a specific order. Each feature that is implemented using the domain specific language will be inherited from the "Feature" class.

The parent class will contain the common virtual methods. Each virtual methods are used to express how each feature will implement each task in a different way. As per in section 4.3.1, there is a lot of events that the system will go through when an order is submitted. Those events will appear in the actual code in these feature class.

Fig. A.20 shows how Limit Order Feature and Market Order Feature have been inherited from their parent feature class. A limit order is added to the order book once it gets matched with its remaining size whereas a market does not get added to the order book even though it does not get matched at OnMatch function.

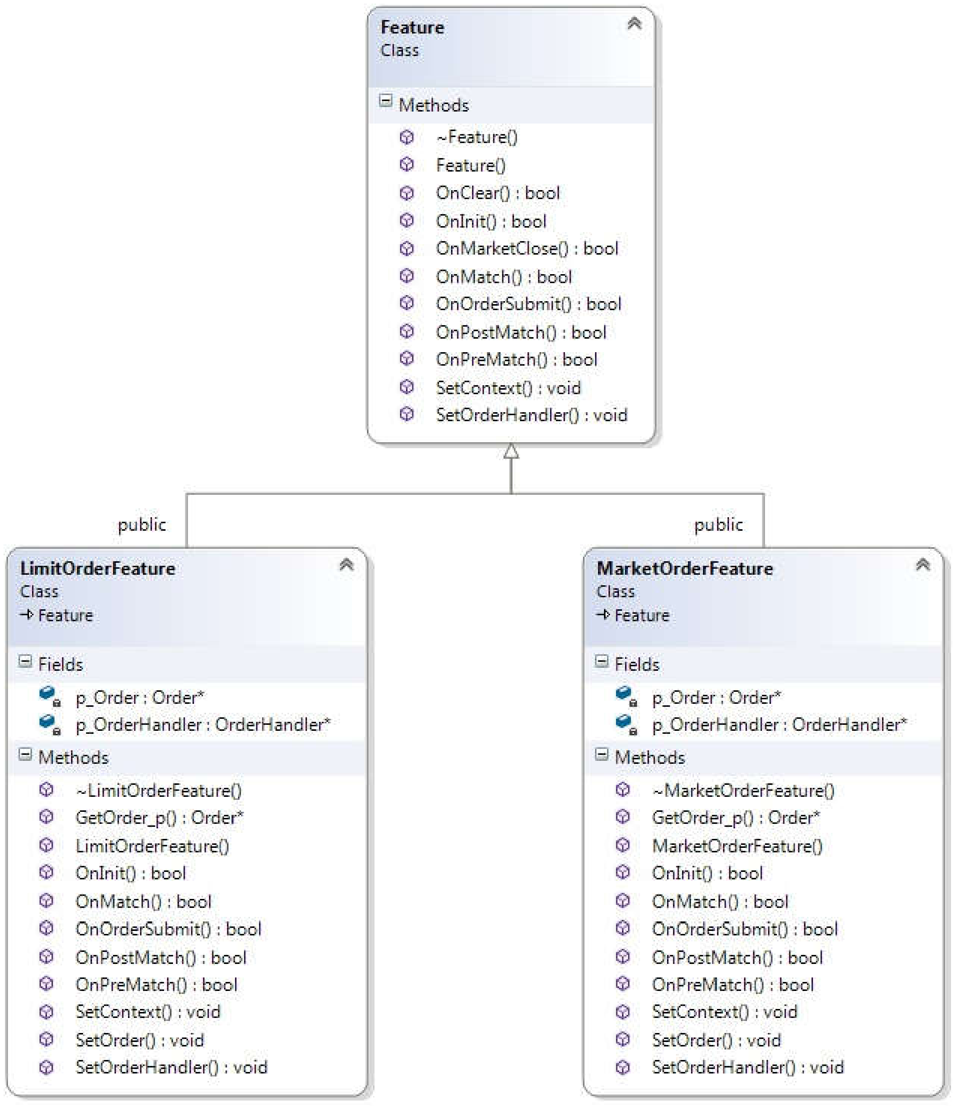


Figure 4.5: Features

Above features have to be included in a feature container. The initialization of features and the main data handler will be the feature container class. When an order is submitted to the controller, it will first check what are the features that are applicable to that specific order. It uses the OnInit function in order to add each applicable feature to a separate list called triggeredFeatureList.

Class diagram of the feature container is shown in Fig. 4.6 . It contains a list of features and helps to load them when they are require by th controller.

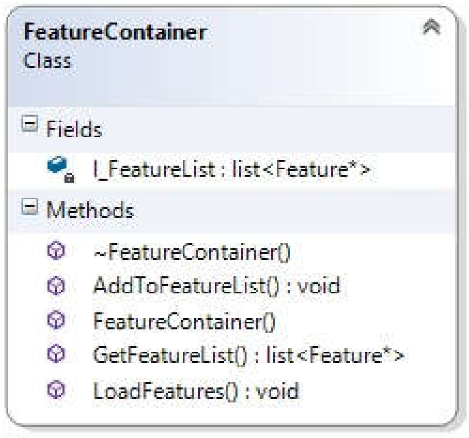


Figure 4.6: Feature Container

An entity can be in the form of different instances. Each instance will have its unique identity and they are inherited from their parent class called "Instance".

As per Fig. 4.7 there can be "Order" instances , "Instrument" instances or even "TradingParameter" instances with unique IDs.As per in section 4.3.1, there can be two types of instances. Base type entity instances are loaded using an external xml where as persistent type entity instances have to be set by the controller as an when they are required.Getter and setter methods get differed depending on the type. All these are handled automatically by the code generator. The user will only have to indicate its type.

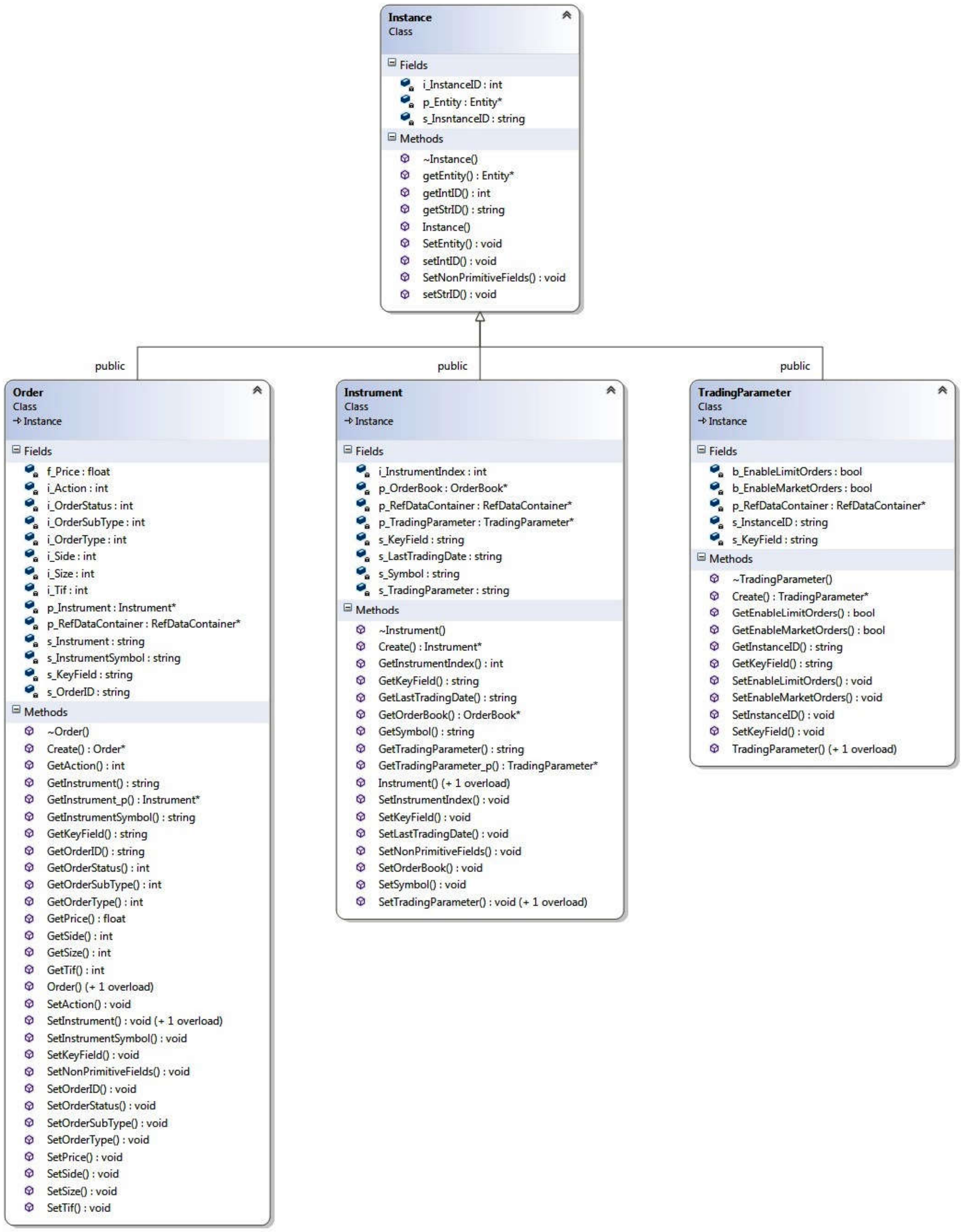


Figure 4.7: Order,Instrument,Trading Parameter Instances

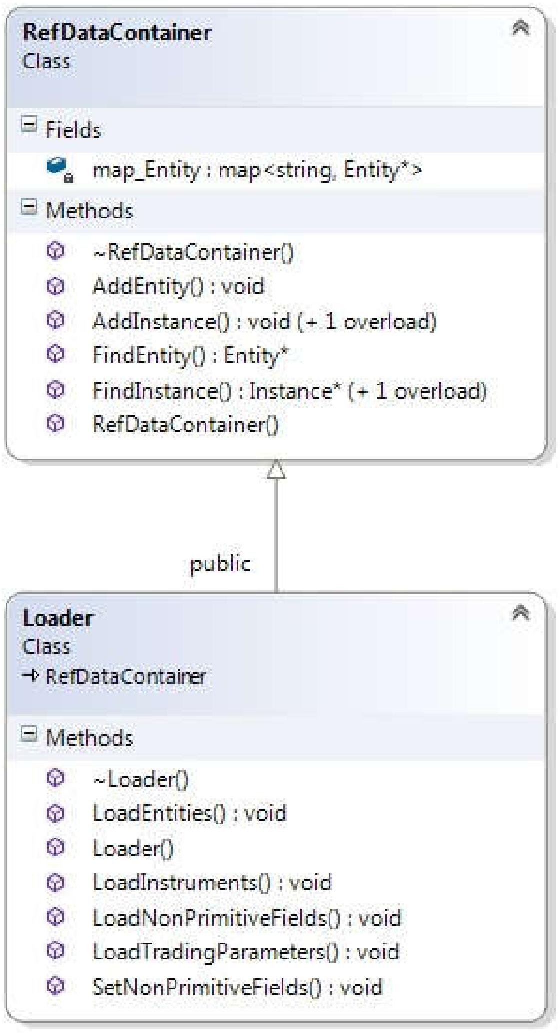
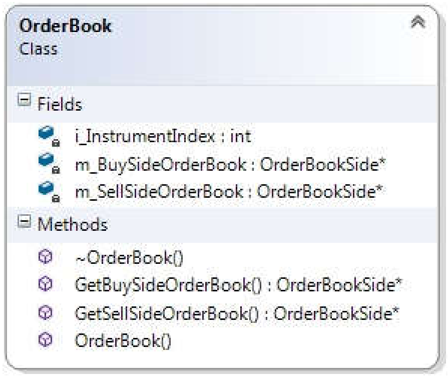


Figure 4.8: Reference Data Container

Reference data has to be stored in some container in order to use in run time. For that purpose RefDataContainer class is used as per Fig. 4.8 . In order to load the reference data from external sources like xml files, a class called "Loader" is used. It has been inherited from the RefDataContainer class.

Loader can load several attributes and it is an automatically generated class by the code generator. When user adds new entity , it will be reflected in the Loader class automatically. It helps to load both entities and its instances.

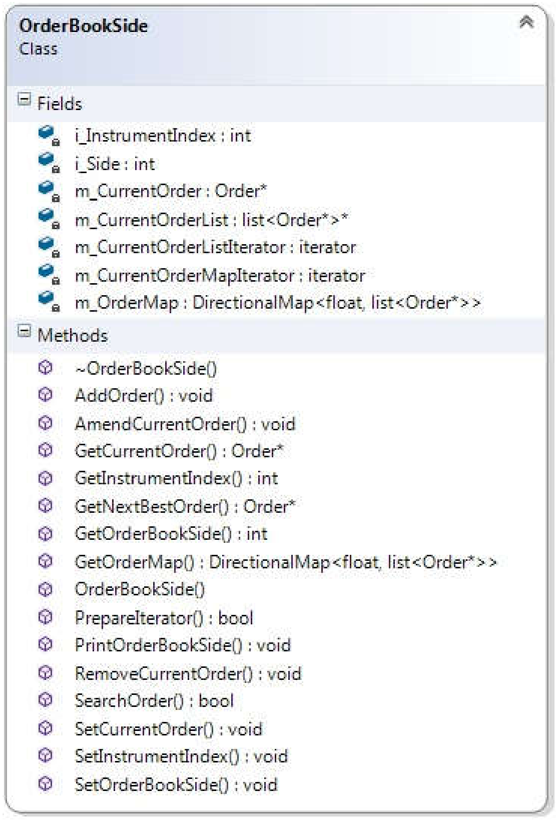
RefDataContainer helps the controller to find instances of each entity as and when they are required by the injected order.


OrderBook
Class

Fields
i_InstrumentIndex : int
m_BuySideOrderBook : OrderBookSide*
m_SellSideOrderBook : OrderBookSide*

Methods
~OrderBook()
GetBuySideOrderBook() : OrderBookSide*
GetSellSideOrderBook() : OrderBookSide*
OrderBook()

Figure 4.9: Order Book


OrderBookSide
Class

Fields
i_InstrumentIndex : int
i_Side : int
m_CurrentOrder : Order*
m_CurrentOrderList : list<Order*>*
m_CurrentOrderListIterator : iterator
m_CurrentOrderMapIterator : iterator
m_OrderMap : DirectionalMap<float, list<Order*>>

Methods
~OrderBookSide()
AddOrder() : void
AmendCurrentOrder() : void
GetCurrentOrder() : Order*
GetInstrumentIndex() : int
GetNextBestOrder() : Order*
GetOrderBookSide() : int
GetOrderMap() : DirectionalMap<float, list<Order*>>
OrderBookSide()
PrepareIterator() : bool
PrintOrderBookSide() : void
RemoveCurrentOrder() : void
SearchOrder() : bool
SetCurrentOrder() : void
SetInstrumentIndex() : void
SetOrderBookSide() : void

Figure 4.10: Order Book Side

Now that all the reference data is ready, there should be a mechanism for an order to be placed after its execution.An order book is a data container which has two sides,namely, buy side order book and sell side order book. As per Fig. 4.9 there are two class members assigned for that.Each side will have separate entries in such a way that buy side has all its orders in the descending order and sell side will be ordered in ascending order. The purpose for that is, the best order in buy side would be the order that has the maximum price and in sell side, the best order would be the order that has the minimum price.

Each order book side has been implemented as per Fig. 4.10 by having a "Directional Map" in it.Directional map is shown in Fig. 4.11 .The use of directional map is to store the orders in ascending and descending order as per the above explanation.

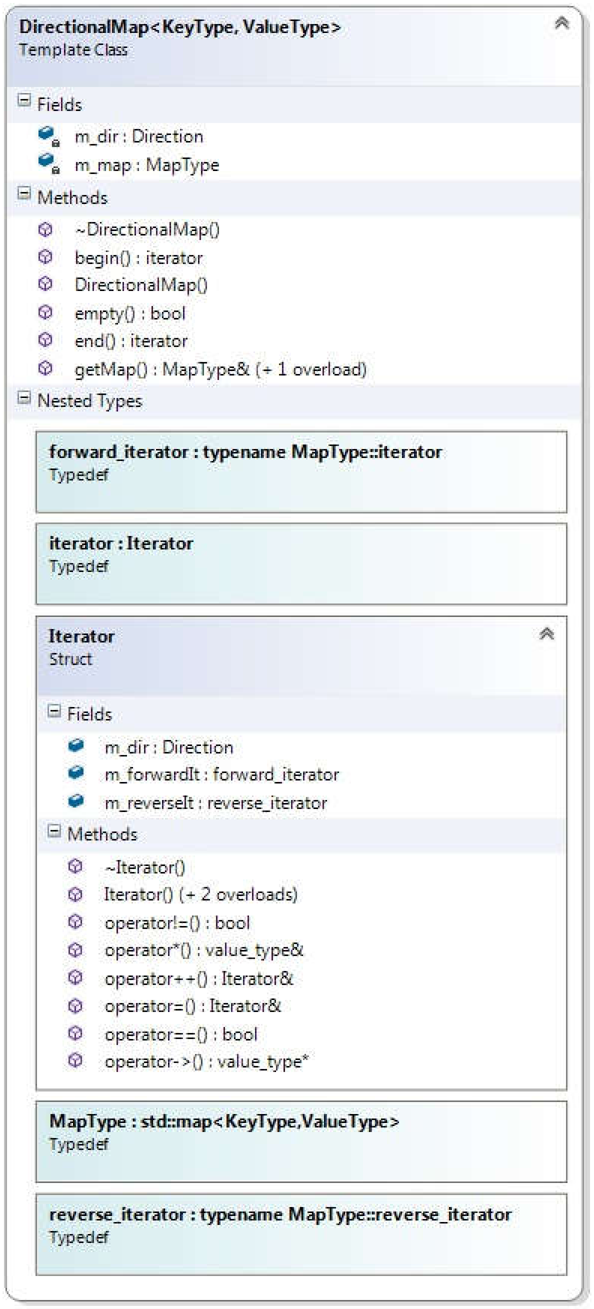

DirectionalMap<KeyType, ValueType>
Template Class
Fields
m_dir : Direction
m_map : MapType
Methods
~DirectionalMap()
begin() : iterator
DirectionalMap()
empty() : bool
end() : iterator
getMap() : MapType& (+ 1 overload)
Nested Types
forward_iterator : typename MapType::iterator
Typedef
iterator : Iterator
Typedef
Iterator
Struct
Fields
m_dir : Direction
m_forwardIt : forward_iterator
m_reverseIt : reverse_iterator
Methods
~Iterator()
Iterator() (+ 2 overloads)
operator!=() : bool
operator*() : value_type&
operator++() : Iterator&
operator=() : Iterator&
operator==() : bool
operator->() : value_type*
MapType : std::map<KeyType,ValueType>
Typedef
reverse_iterator : typename MapType::reverse_iterator
Typedef

Figure 4.11: Directional Map

The direction of the map has to be defined when order book class is created.After that directional map will handle all its iterator and standard maps itself.

OrderBookSide is one of the main classes in this design. It has several methods to perform actions upon injected orders.Mainly this controller will have following action when an order is submitted.

1. **New Order** : - A new order is an order which could be executed with the orders in the order book or even get added to the order book if the feature permits to to so. For an example, a leave order will be added to the order book after its execution whereas market orders will get expired if it does not fully matched with an order that exists in the counter order book side.

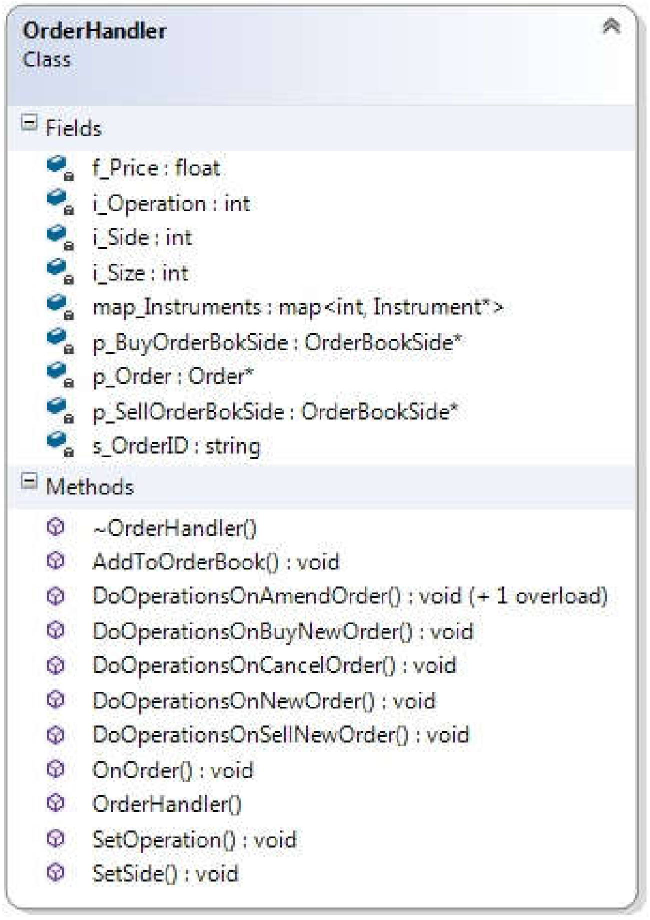
   When a new order is submitted, the controller will find its best counter order and tries to match it. For that the directional map will help to get the next best order by providing the correct map iterator.

   When the order is partially executed, that will be added to the order book based on its price. Price will be the key to the map. The map will have separate lists for each price point. The list will make sure that the price priority and time priority of each order is maintained correctly. More priority will be given to best price and for the order that came early when choosing the next best order for match.

2. **Amend Order** :- The users could amend the existing orders in any order book side by providing its order ID. When such request comes, the controller will first find that order and then amend the price or size based on the request.If the order price is different than the original order price,it will be removed from the existing list and put in the list with the new price point. The time priority will get changed in this case. If order size is less than the original size, the size will get amended without loosing the time priority. If the size is greater than the original order size, the time priority will get changed.All are handled by the controller.

3. **Cancel Order** :- The user can cancel an order by providing its respective order ID.Then the order will be found from the directional map and get removed. If the corresponding order list is empty , the respective list will be removed too.

The above mentioned operations are handled by the Order Handler class as per in Fig. 4.12. The operations on each order book side have been handled separately by considering their data structure.

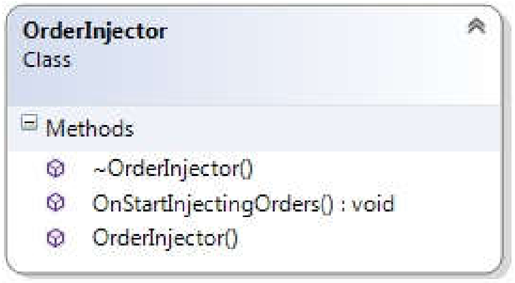


```
f_Price : float
i_Operation : int
i_Side : int
i_Size : int
map_Instruments : map<int, Instrument*>
p_BuyOrderBokSide : OrderBookSide*
p_Order : Order*
p_SellOrderBokSide : OrderBookSide*
s_OrderID : string
```

```
~OrderHandler()
AddToOrderBook() : void
DoOperationsOnAmendOrder() : void (+ 1 overload)
DoOperationsOnBuyNewOrder() : void
DoOperationsOnCancelOrder() : void
DoOperationsOnNewOrder() : void
DoOperationsOnSellNewOrder() : void
OnOrder() : void
OrderHandler()
SetOperation() : void
SetSide() : void
```

Figure 4.12: Order Handler

In order to inject orders to the main system or the controller, an Order Injector class is used as per Fig. 4.13 which will provides a callback function to the controller.That was designed using the "observer" design pattern.



```
~OrderInjector()
OnStartInjectingOrders() : void
OrderInjector()
```

Figure 4.13: Order Injector

In order to provide a visible output as to what has been happened in the controller, a logger class is used. Singleton design pattern is used in order to implement the logger, which will have only one instance to log the event for the whole controller. The output is written to a file so that the users could get an idea on how the outputs are generated from the controller for each feature.

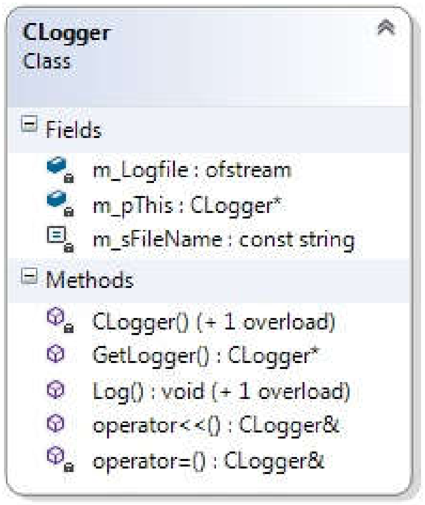


Figure 4.14: Logger

4.3.11 Summary of implementation

The controller consists of two components.Namely auto generated codes and the core. Auto generated codes are generated from the code generator and they are generated using the domain specific language which are defined using Antlr4 grammar and then converted to C++ using StringTemplates.

It is advised not to change any auto generated code segments in order to maintain the controller in a proper way. Each feature must be defined independently and could be used in the controller without any other input from any other feature.

The domain specific language could be even converted to languages other than C++ because of its design and use of StringTemplates.

Two separate applications were developed for the controller in C++ language and the code generator in Java language.Therefore in the evaluation, first the code generator behavior is evaluated and then the controller behavior is evaluated in chapter 5.0

CHAPTER 5

SYSTEM EVALUATION

5.1 Overview

After carrying out a research , the results should be evaluated and checked whether the results are accurate and valid.That is a very important component of any research and justification is a must.

In this section, the research which I have carried out will be evaluated and justified to be true. The main implementation of this research is the code generator. The semantic model has been developed along with the main controller. The controller could be used here to validate and evaluate the results. It is an added advantage of this design.

5.2 Evaluation of Code Generator

As I already mentioned in the previous section,the code generator could be evaluated using the main controller itself. The below section will describe the inputs are injected to the system in order to evaluate the results.

For this evaluation, some important researches were refereed in order to get ideas on how to test and evaluate this research accurately.One of the researches were done by Ludvig which describes how to test domain specific languages [42] was analyzed in this regard.They have describes a concept called software product lines [27]. Not like a one off development, the software product lines are developed like a series of software products.Model driven testing scenarios has also been studied in order to carry out this research [31] [32].

5.3 How to provide inputs

The inputs are injected using the xml files. The controller will read the input data and process using the auto generated code files.The auto generated code files which are generated by our code generator are integrated before running the main controller.The inputs are injected as per below xml format.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Instances>
  <Instance>
    <instrumentSymbol>10YR_CR02</instrumentSymbol>
    <instrument>23</instrument>
    <orderType>1</orderType>
```

```
      <size>100</size>
      <price>12.5</ price >
      < tif >1</ tif >
      <side>1</side>
      <orderID>1q2yu4</orderID>
      <orderSubType>1</orderSubType>
      <action>1</action>
 </ Instance >
  <Instance>
      <instrumentSymbol>10YR_CR02</instrumentSymbol>
      <instrument>23</instrument>
      <orderType>1</orderType>
      <size>200</size>
      <price>13.7</ price >
      < tif >1</ tif >
      <side>1</side>
      <orderID>2t6as8</orderID>
      <orderSubType>1</orderSubType>
      <action>1</action>
 </ Instance >
 <Instance>
      <instrumentSymbol>10YR_CR02</instrumentSymbol>
      <instrument>23</instrument>
      <orderType>1</orderType>
      <size>100</size>
      <price>13.5</ price >
      < tif >1</ tif >
      <side>2</side>
      <orderID>3r8gr9</orderID>
      <orderSubType>1</orderSubType>
      <action>1</action>
 </ Instance >
<Instance>
      <instrumentSymbol>10YR_CR02</instrumentSymbol>
      <instrument>23</instrument>
      <orderType>1</orderType>
      <size>150</size>
      <price>12.5</ price >
      < tif >1</ tif >
      <side>2</side>
      <orderID>4p3tr6</orderID>
      <orderSubType>1</orderSubType>
      <action>1</action>
 </ Instance >
  <Instance>
      <instrumentSymbol>10YR_CR02</instrumentSymbol>
      <instrument>23</instrument>
      <orderType>1</orderType>
      <size>200</size>
      <price>11.75</ price>
      < tif >1</ tif >
      <side>2</side>
      <orderID>5y9ew6</orderID>
      <orderSubType>1</orderSubType>
      <action>1</action>
 </ Instance >
 <Instance>
     <instrumentSymbol>10YR_CR02</instrumentSymbol>
     <instrument>23</instrument>
     <orderType>1</orderType>
     <size>100</size>
     <price>10.25</ price>
```

```
    < tif >1</ tif >
    <side>2</side>
    <orderID>6q7ui7</orderID>
    <orderSubType>1</orderSubType>
    <action>1</action>
  </ Instance >
  <Instance>
    <instrumentSymbol>10YR_CR02</instrumentSymbol>
    <instrument>23</instrument>
    <orderType>1</orderType>
    <size>100</size>
    <price>9.50</ price >
    < tif >1</ tif >
    <side>2</side>
    <orderID>6q7ui7</orderID>
    <orderSubType>1</orderSubType>
    <action>2</action>
  </ Instance >
  <Instance>
    <instrumentSymbol>10YR_CR02</instrumentSymbol>
    <instrument>23</instrument>
    <orderType>1</orderType>
    <size>100</size>
    <price>15.5</ price >
    < tif >1</ tif >
    <side>2</side>
    <orderID>6q7ui7</orderID>
    <orderSubType>1</orderSubType>
    <action>2</action>
  </ Instance >
  <Instance>
    <instrumentSymbol>10YR_CR02</instrumentSymbol>
    <instrument>23</instrument>
    <orderType>2</orderType>
    <size>100</size>
    <price>10.0</ price >
    < tif >1</ tif >
    <side>2</side>
    <orderID>7j5aq6</orderID>
    <orderSubType>1</orderSubType>
    <action>1</action>
  </ Instance >
  <Instance>
    <instrumentSymbol>10YR_CR02</instrumentSymbol>
    <instrument>23</instrument>
    <orderType>1</orderType>
    <size>100</size>
    <price>15.5</ price >
    < tif >1</ tif >
    <side>2</side>
    <orderID>6q7ui7</orderID>
    <orderSubType>1</orderSubType>
    <action>3</action>
  </ Instance >

</ Instances >
```

5.4 Evaluating the outputs

5.4.1 How to evaluate the outputs

The stock exchange system that has been developed supports 3 basic operations.Namely;

1. Putting a new order.
2. Amend an existing order.
3. Cancel an existing order.

Above basic operations should be validated using the controller.The test cases are manually created for each operation as per following tables.Apart from the manual test cases another tool has been written in order to validate the results automatically.

5.4.2 Automated validator

This automated validator has been written in Java language in order provide same inputs and get the results so that we could compare the results with the actual outcomes taken from our controller. The validator accepts two csv files for orders and instruments whereas actual system accepts xml files for order input.Validator has been implemented without any feature interaction and the design is completely different. Similar functionality has been implemented in both controller and in validator, so that the comparison is valid and done under same conditions along with same inputs.

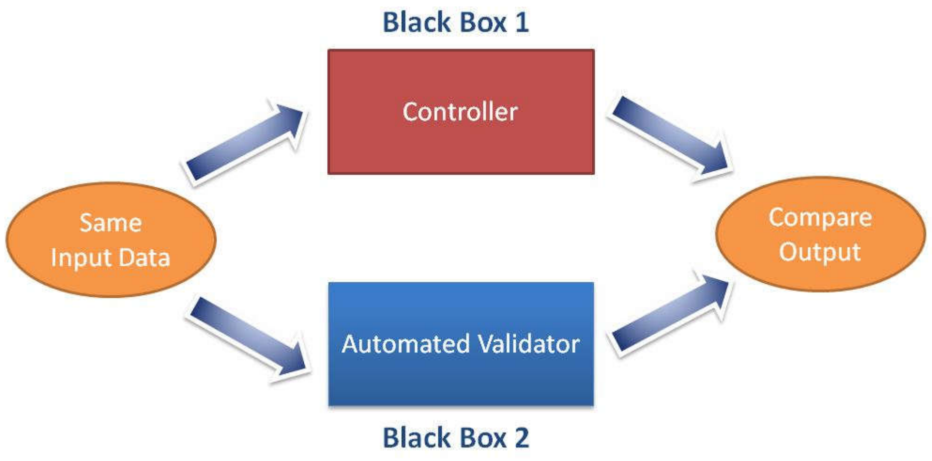


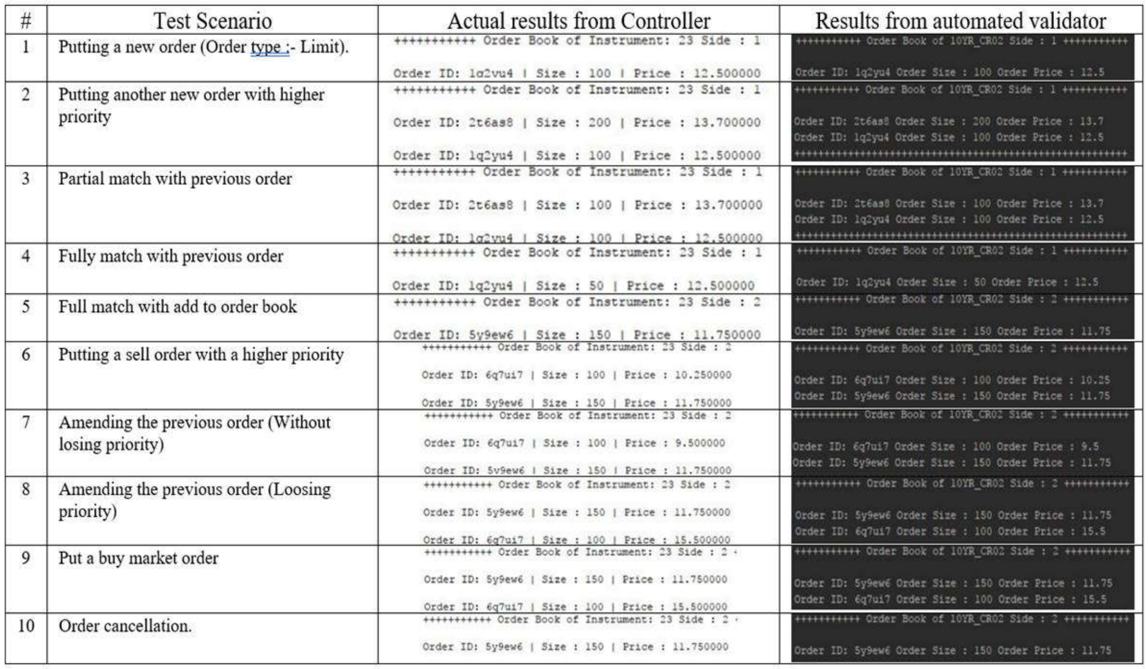
Figure 5.1: Evaluation methodology

Actually this validator has been developed prior to the research in order to identify as to how the controller should break into separate features .As per Fig. 5.1 if both the results are same we can conclude that our controller has been implemented accurately.

After calculating the results manually and using the above mentioned automated test results we can have a comparison as per in section 5.4.3 . When evaluating the results we could use this comparison as a proof.

5.4.3 Comparison between manually calculated results,automated test results and actual results

Below in Fig. 5.2 has a summary of results obtained by carrying out several test cases.



#	Test Scenario	Actual results from Controller	Results from automated validator
1	Putting a new order (Order type :- Limit).	+++++++++++ Order Book of Instrument: 23 Side : 1 Order ID: lq2yu4 | Size : 100 | Price : 12.500000	+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++ Order ID: lq2yu4 Order Size : 100 Order Price : 12.5
2	Putting another new order with higher priority	+++++++++++ Order Book of Instrument: 23 Side : 1 Order ID: 2t6as8 | Size : 200 | Price : 13.700000 Order ID: lq2yu4 | Size : 100 | Price : 12.500000	+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++ Order ID: 2t6as8 Order Size : 200 Order Price : 13.7 Order ID: lq2yu4 Order Size : 100 Order Price : 12.5 ++
3	Partial match with previous order	+++++++++++ Order Book of Instrument: 23 Side : 1 Order ID: 2t6as8 | Size : 100 | Price : 13.700000 Order ID: lq2yu4 | Size : 100 | Price : 12.500000	+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++ Order ID: 2t6as8 Order Size : 100 Order Price : 13.7 Order ID: lq2yu4 Order Size : 100 Order Price : 12.5 ++
4	Fully match with previous order	+++++++++++ Order Book of Instrument: 23 Side : 1 Order ID: lq2yu4 | Size : 50 | Price : 12.500000	+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++ Order ID: lq2yu4 Order Size : 50 Order Price : 12.5
5	Full match with add to order book	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
6	Putting a sell order with a higher priority	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 6q7ui7 | Size : 100 | Price : 10.250000 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 6q7ui7 Order Size : 100 Order Price : 10.25 Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
7	Amending the previous order (Without losing priority)	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 6q7ui7 | Size : 100 | Price : 9.500000 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 6q7ui7 Order Size : 100 Order Price : 9.5 Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
8	Amending the previous order (Loosing priority)	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000 Order ID: 6q7ui7 | Size : 100 | Price : 15.500000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75 Order ID: 6q7ui7 Order Size : 100 Order Price : 15.5
9	Put a buy market order	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000 Order ID: 6q7ui7 | Size : 100 | Price : 15.500000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75 Order ID: 6q7ui7 Order Size : 100 Order Price : 15.5
10	Order cancellation.	+++++++++++ Order Book of Instrument: 23 Side : 2 Order ID: 5y9ew6 | Size : 150 | Price : 11.750000	+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++ Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75

Figure 5.2: Summary of test results

Each of above test cases exhibits a different behavior of main controller.The system evaluation could be based on the above obtained results. The details of each test case has been included in Appendix A.

5.4.4 Evaluation of results

In order to evaluate the results, information extracted from the log file is being used. The log file is generated by the controller along with the time of each event happened. Each of above scenarios could be run and results could be evaluated with the expected outcomes.Expected outcomes have been both automatically calculated using an external validator and manually calculated .If there is no difference between the results generated by the controller and the expected results , we could validate that the results are accurate.

The results of each of the above 10 scenarios can be evaluated as follows;

- As per Table 5.1 when putting the new order , there is no counter order to get matched and also it is a limit order so it directly added to the order book.
- When putting the second order at a higher price than the previous order it gets added to the top of the order book as per in Table 5.2
- A partial match happens when a sell order is put for a matching quantity of 100 and the existing order size gets reduced by 100 as per Table 5.3
- With putting an order with size 150, order ID 2t6as8 gets fully matched as per Table 5.4. This could be further evaluated by performing another full match in such a way that the remaining quantity of the sell order gets added to the sell side of the order book as per the Table 5.5
- Now we add another sell order with a higher priority (which is priced at 10.25) . In sell side,higher priority means lower price and in the buy side , higher priority is given to the orders with high prices. It reflects in Table 5.6
- Now that new order functionality is being evaluated, we could move on to the next functionality which is order amendment. When we change the order price to a lower value , the priority won't change as per Table 5.7.
- Again when we change the price of order 6q7ui7 from 9.5 to 15.5, it's priority gets reduced as per Table 5.8. Therefore order amendment is also evaluated and then moving on to market orders.
- A market order is an order which gets expired when it is not matched with the existing orders in the order book. So either it gets fully matched or partially

matched otherwise it will get expired as per Table 5.9

- Finally order cancellation functionality is being evaluated as per Table 5.10 and it could be seen that each and every above 10 scenarios are validated with the respective figures extracted from its log file.

With the use of the manually calculated expected results and the automated test results we could conclude that the research is a success as the results are accurate.It means that the automatically generated code segments from our code generator have been properly worked.

This evaluation has been done considering two features which are Limit Order Feature and Market Order Feature. It could be even modeled using more than two features .

CHAPTER 6

CONCLUSION

6.1 Contribution

This research has been a success for various reasons. Firstly , a massive stock exchange system has been decoupled or modeled in terms of a set of features.If a particular feature is not required, the feature could be easily decoupled from all the code segments with minimum manual intervention.

Secondly the code is being automatically generated which supports even business analysts to add business logic themselves using the language I have introduced.This could be further enhanced by introducing graphical user interfaces which I have been mentioned in section 6.3 as future improvements to the system.

Thirdly ,the end outcome or the code generator has provided an efficient and clean way to manage the codes in the newly introduced feature oriented software development methodology. Before compiling the code, the code generator will generate the required files in accordance with the feature list required to implement a solution.There will not be any code segment which is not used or functioning inside the main code hereafter. The code will be clean and easy to understand even by a new developer.

Many languages and technologies were used in carrying out this research. Mainly the code generator has been developed using Java,StringTemplates and Antlr . The code generator has been developed using another language (C++) in order to exhibit that the output of the code generator could be provided in any user defined language. The auto generated code could be in any language and it is a real advantage of this methodology.

6.2 Study limitations

This research has been limited to the stock exchange systems domain. The results obtained in section 5 is applicable in stock exchange systems and also each feature must not interact or interfere with one another. Feature interaction has been presented in the next section under future work.

6.3 Future work

This research mainly focuses on a code generator which provides better structuring and feature oriented capabilities to stock exchange based systems. This research could be improved further if the code base is changed to a graphical model or visual programming language (VPL) model. In the current implementation, the person who wants to enter the logics ,has to code according to the given syntaxes .If the coding method is changed to a visual programming model , the users would be able to drag and drop the functionality modules and implement the system which will replace the Antlr parser in the current implementation.If this is converted to a such model ,it could be even used to promote the product to customers mentioning that they will also be capable of introducing changes by themselves using the graphical drag and drop coding model.

Further there is a limitation which has not been addressed by this research. That is, how to handle the features which are dependent on another feature.This kind of feature interaction problem has been evaluated by T. Bowen and his team in their research on how to solve feature interaction in telecommunication systems [19] .How to detect such interactions have been described by Sven Apel and his team,in their research [29].They have described on how to separate each concern and how to detect dependencies and interactions between features.An efficient way to detect the feature interaction has also been explained by Cynthia Disenfeld and team [33].Even there are researches done in order to find out how much these features are inter-related .One such research is done by Joanne and team [34] The cross functionality has not been considered in this research and as a future improvement , that function could be added. The feature interactions have to be taken into account and it should be implemented with more care because the combined feature should give the desired result without causing any error in output.

Bibliography

[1] Sven Apel, Christian Kastner,"An Overview of Feature Oriented Software Development", in Journal of Object Technology, vol. 8, no. 4, pages 1-36,July - August 2009.

[2] Feature-oriented programming From Wikipedia [Online] Available https://en.wikipedia.org/wiki/Feature-oriented_programming

[3] Eth Zurich,Chair of Engineering,"An Overview of Feature-Oriented Software Development",Journal of Object Technology Vol 8 ,Augest 2009

[4] K. Kang, S. Kim, J. Lee, K. Kim, G. Kim, and E. Shin." FORM A Feature-Oriented ReuseMethod with Domain-Specific Reference Architectures", Annals of Software Engineering, pages 143 - 168, 1998.

[5] . Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90- TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[6] A. Classen, P. Heymans, and P. Schobbens. "What is in a Feature- A Requirements Engineering Perspective", In Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), volume 4961 of Lecture Notes in Computer Science, pages 16 to 30. Springer Verlag , 2008.

[7] J. Bosch. Design and Use of Software Architectures - Adopting and Evolving a Product-Line Approach. ACM Press / Addison-Wesley, 2000.

[8] K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[9] Sven Apel, Thomas Leich, Marko Rosenm uller, Gunter Saake, "FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming", in: Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE), Vol. 3676 of LNCS, Springer-Verlag, 2005

[10] Sven Apela, Sergiy Kolesnikova, J.org Liebiga, Christian K.astnerb, Martin Kuhlemannc, Thomas Leichd,"Access Control in Feature-Oriented Programming",Elsevier,August 11, 2010

[11] SVEN APEL,DELESLEY HUTCHINS,"A Calculus for Uniform Feature Composition,ACMTransactions on Programming Languages and Systems", Vol. 32, No. 5,Article 19, Publication date:May 2010.

[12] Don Batory, Member, IEEE, Jacob Neal Sarvela, Student Member, IEEE, and Axel Rauschmayer, Student Member, IEEE,Scaling Step-Wise Refinement,IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 6, JUNE 2004

[13] A. Colyer and A. Clement. "Large-Scale AOSD for Middleware",Proceedings of the 3rd international conference on Aspect-oriented software development,Pages 56-65 , 2004

[14] R. Laddad,"AspectJ in Action Practical Aspect-Oriented Programming.",Manning Publications Co.Greenwich, CT, USA , 2003

[15] D Batory and S O Malley, "The Design and Implementation of Hierarchical Software Systems with Reusable Components" , ACM Trans. Software Eng. Methodology, Oct. 1992.

[16] S. Apel, T. Leich, G. Saake, Aspectual Feature Modules, IEEE Transactions on Software Engineering (TSE) 34 (2) (2008) 162 - 180.

[17] S. Apel, C. K.astner, C. Lengauer, "FeatureHouse: Language-Independent, Automated Software Composition", in: Proceedings of the International Conference on Software Engineering (ICSE), IEEE Computer Society, 2009, pp. 221 - 231.

[18] R. Lopez-Herrejon, D. Batory, W. Cook, Evaluating Support for Features in Advanced Modularization Technologies, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Vol. 3586 of LNCS, Springer-Verlag, 2005, pp. 169 -194

[19] T. Bowen, F. Dworack, C. Chow, N. Griffeth, and G. Herman Y.J. Lin. "The Feature Interaction Problem in Telecommunications Systems", In Proceedings of the International Conference on Software Engineering for Telecommunication Switching Systems (SETSS) , pages 59 - 62 ,IEEE CS Press, 1989.

[20] Sven Apel; Florian Heidenreich; Christian Kastner; Marko Rosenmuller , "Third International Workshop on Feature-Oriented Software Development",15th International Software Product Line Conference,Pages: 337 - 338,2011

[21] Keisuke Yano; Akihiko Matsuo,"Labeling Feature-Oriented Software Clusters for Software Visualization Application",Asia-Pacific Software Engineering Conference (APSEC),Pages: 354 - 361,2015

[22] J. K. Muller,"Feature - Oriented Software Structuring" Computer Software and Applications Conference, COMPSAC '97. Proceedings, The Twenty-First Annual International,Pages: 552 - 555,1997

[23] Mehran Kavand; Saeed Paarsa; Ahmad Faraahi,"A context-independent feature-oriented software development approach",6th International Conference on Computer Science & Education (ICCSE),Pages: 1115 - 1122 ,2011

[24] M. Mezini and K. Ostermann. Variability Management with Feature-Oriented Programming and Aspects. In Proceedings of the International Symposium on Foundations of Software Engineering, pages 127-136.ACM Press, 2004.

[25] Christian Kastner; Thomas Thum; Gunter Saake; Janet Feigenspan; Thomas Leich; Fabian Wielgorz; Sven Apel,"FeatureIDE: A tool framework for feature-oriented software development",IEEE 31st International Conference on Software Engineering,Pages: 611 - 614,2009

[26] C. Marimuthu; K. Chandrasekaran,"Feature-Oriented Domain Analysis Framework for Energy-Aware Self-Adaptive Software",IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData),Pages: 773 - 776,2016

[27] Beatriz Perez Lamancha; Oscar Diaz; Maider Azanza; Macario Polo,"Software product line testing: A feature oriented approach",IEEE International Conference on Industrial Technology,Pages: 298 - 305,2012

[28] Roberto E. Lopez-Herrejon; Leticia Montalvillo-Mendizabal; Alexander Egyed"From Requirements to Features: An Exploratory Study of Feature-Oriented Refactoring",15th International Software Product Line Conference,Pages: 181 - 190, 2011

[29] Sven Apel; Wolfgang Scholz; Christian Lengauer; Christian Kastner,"Detecting Dependences and Interactions in Feature-Oriented Design",,IEEE 21st International Symposium on Software Reliability Engineering,Pages: 161 - 170,2010

[30] Herbert Prahofer; Daniela Rabiser; Florian Angerer; Paul GrÃijnbacher; Peter Feichtinger,"Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support",,IEEE 14th International Conference on Industrial Informatics (INDIN),Pages: 1218 - 1223,2016

[31] P. Sochos; M. Riebisch; I. Philippow,"The feature-architecture mapping (FArM) method for feature-oriented development of software product lines",13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06),Pages: 9 pp. - 318,2006

[32] Sven Apel; Dirk Beyer,"Feature cohesion in software product lines: an exploratory study",2011 33rd International Conference on Software Engineering (ICSE),Pages: 421 - 430,2011

[33] Cynthia Disenfeld; Ioanna Stavropoulou; Julia Rubin; Marsha Chechik,"FPH: Efficient Detection of Feature Interactions through Non-Commutativity",2017

IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C),Pages: 225 - 225,2017

[34] Joanne M. Atlee; Uli Fahrenberg; Axel Legay,"Measuring Behaviour Interactions between Product-Line Features",IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering,Pages: 20 - 25,2015

[35] R. Piantoni; C. Stancescu,"Implementing the Swiss Exchange trading system",Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing,Pages: 309 - 313,1997

[36] Bikash Dhakal; Manoj Kumar Gupta,"A system model of online trading system for Nepal Stock Exchange",International Conference on Computing for Sustainable Global Development (INDIACom),Pages: 367 - 372,2014

[37] K R Neeraj; P S Janardhanan; Anu Bonia Francis; Reena Murali,"A domain specific language for business transaction processing",IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES),Pages: 1 - 7,2017

[38] Van Deursen, Arie, Paul Klint, and Joost Visser," Domain-specific languages An annotated bibliography", ACM Sigplan Notices,pages 26-36,

[39] Mernik, Marjan, Jan Heering, and Anthony M. Sloane ,"When and how to develop domain specific languages",ACM computing surveys (CSUR),pages 316-344,2000

[40] Danyang Cao; Donghui Bai,"Design and implementation for SQL parser based on ANTLR",2010 2nd International Conference on Computer Engineering and Technology, Volume: 4,Pages: V4-276 - V4-279,2010

[41] Yueming Zhao; Teng Wang; Xiaoyu Ni; Xin'an Wang; Zheng Xie,"Syntactic Representation Transformation in Operator Design Method Based on ANTLR Tool",IEEE 12th International Conference on Computer and Information Technology,Pages: 115 - 118,2012

[42] Ludvig Kihlman,"A test model for domain-specific language development",2017 9th Computer Science and Electronic Engineering (CEEC),Pages: 207 - 212,2017

[43]] P. W. Burke and P. Sweany, "Automatic Code Generation Through Model-Driven Design", University of North Texas, Denton, Texas, October 23, 2007.

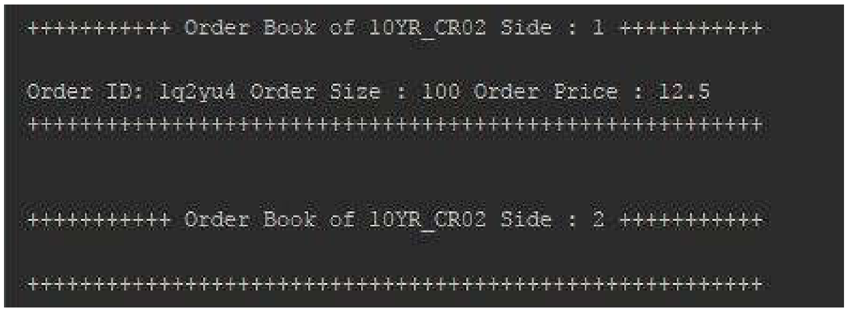
Appendix A

DETAILED TEST RESULTS

1. **Putting a new order (Order type :- Limit)**

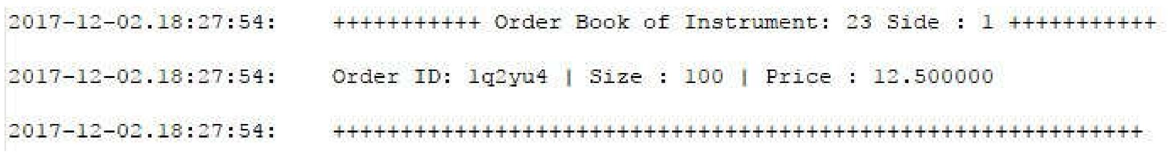
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
1q2yu4	100	12.500000			

Table A.1: Putting a limit new order



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

Order ID: 1q2yu4 Order Size : 100 Order Price : 12.5
++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.1: Automated test result for test case in Table 5.1



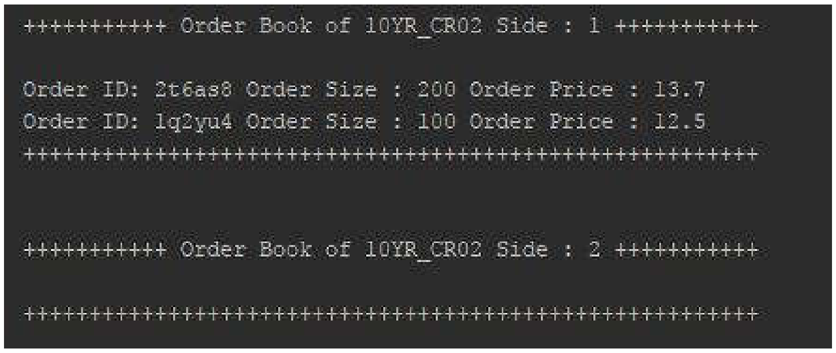
```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 1 +++++++++++

2017-12-02.18:27:54:    Order ID: 1q2yu4 | Size : 100 | Price : 12.500000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.2: Actual result extracted from log file for test case in Table 5.1

2. **Putting another new order with higher priority(Order type :- Limit)**

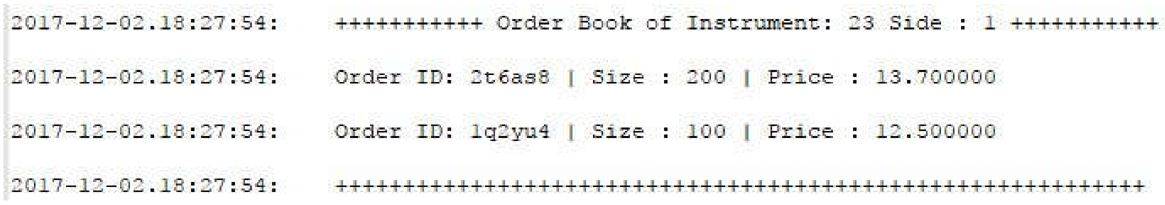
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
2t6as8	200	13.700000			
1q2yu4	100	12.500000			

Table A.2: Putting limit new order with higher priority



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

Order ID: 2t6as8 Order Size : 200 Order Price : 13.7
Order ID: 1q2yu4 Order Size : 100 Order Price : 12.5
++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.3: Automated test result for test case in Table 5.2



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 1 +++++++++++

2017-12-02.18:27:54:    Order ID: 2t6as8 | Size : 200 | Price : 13.700000

2017-12-02.18:27:54:    Order ID: 1q2yu4 | Size : 100 | Price : 12.500000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.4: Actual result extracted from log file for test case in Table 5.2

3. **Partial match with previous order**

Here we put a sell order for the same instrument with size 100 and price 13.5 . The resulting order book should have following figures.

Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
2t6as8	100	13.700000			
1q2yu4	100	12.500000			

Table A.3: Putting sell order for partial match

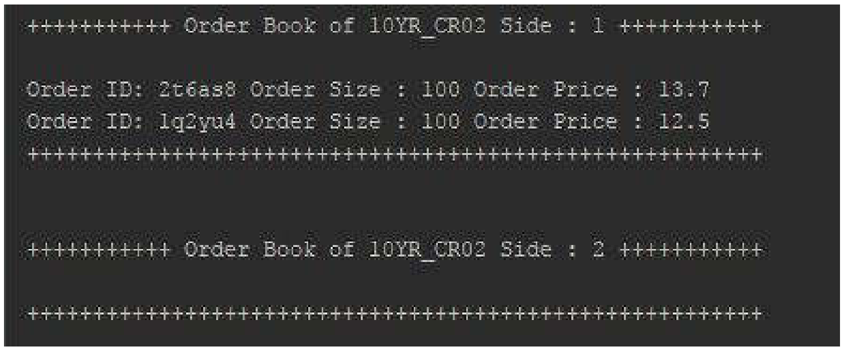

```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

Order ID: 2t6as8 Order Size : 100 Order Price : 13.7
Order ID: 1q2yu4 Order Size : 100 Order Price : 12.5
+++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.5: Automated test result for test case in Table 5.3

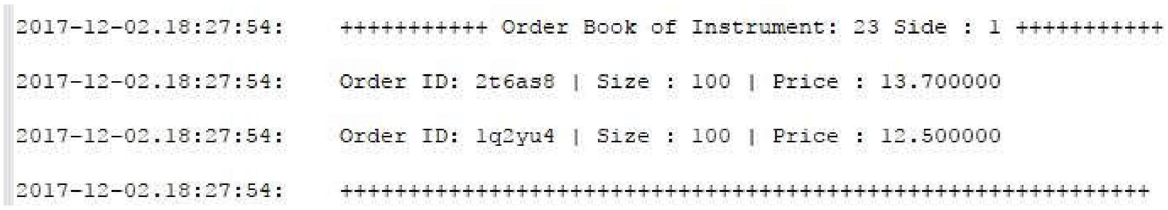

```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 1 +++++++++++

2017-12-02.18:27:54:    Order ID: 2t6as8 | Size : 100 | Price : 13.700000

2017-12-02.18:27:54:    Order ID: 1q2yu4 | Size : 100 | Price : 12.500000

2017-12-02.18:27:54:    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

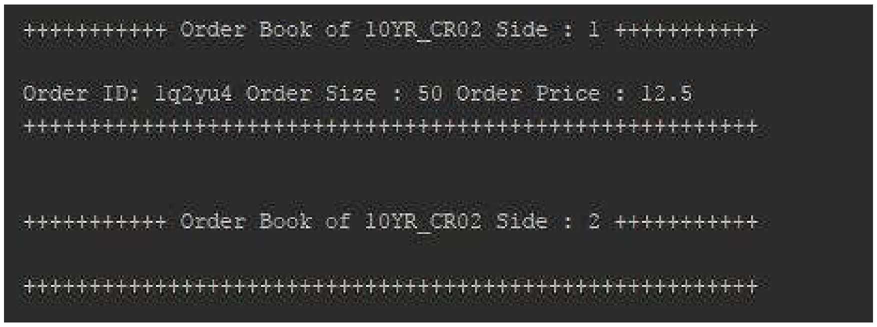
Figure A.6: Actual result extracted from log file for test case in Table 5.3

4. **Fully match with previous order**

   Now we put a sell order with size of 150 at a price of 11.5 and the resulting order book should have following figures

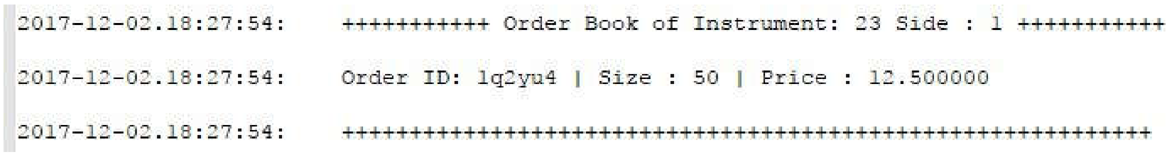
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
1q2yu4	50	12.500000			

Table A.4: Putting a sell order for full match



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

Order ID: 1q2yu4 Order Size : 50 Order Price : 12.5
+++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.7: Automated test result for test case in Table 5.4



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 1 +++++++++++
2017-12-02.18:27:54:    Order ID: 1q2yu4 | Size : 50 | Price : 12.500000
2017-12-02.18:27:54:    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

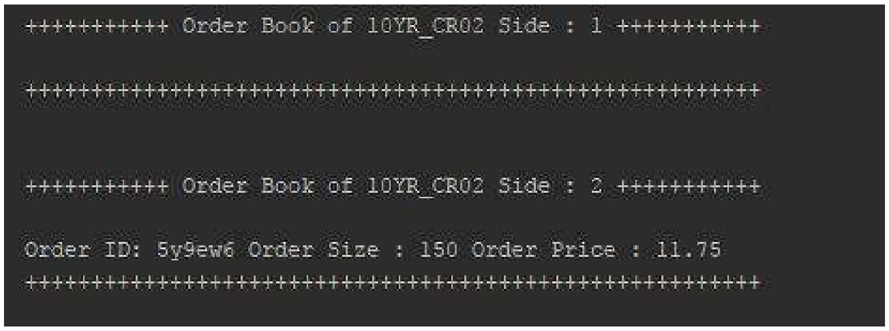
Figure A.8: Actual result extracted from log file for test case in Table 5.4

5. **Full match with add to book**

   Here we can test a full match of a order in its counter order book side and the remaining quantity being added to the respective side. An order having size of 200 is put at a price of 11.75. Resulting orderbook can be seen as per table 5.5.

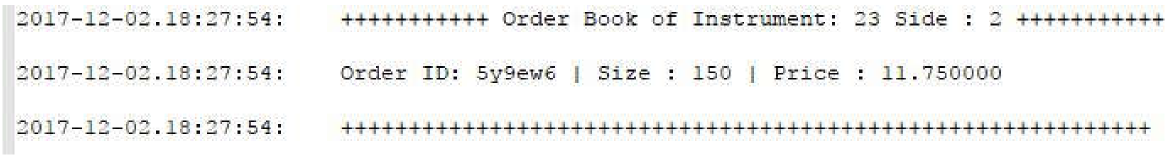
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			11.750000	150	5y9ew6

Table A.5: Putting a sell order for full match and adding remaining quantity to order book



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.9: Automated test result for test case in Table 5.5



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

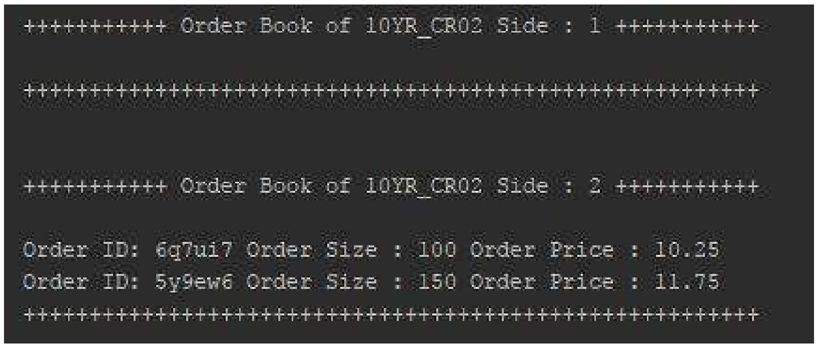
Figure A.10: Actual result extracted from log file for test case in Table 5.5

6. **Putting a sell order with a higher priority**

   Here an order is placed which is a sell order. The order has a higher priority than exisiting orders in the current order book.

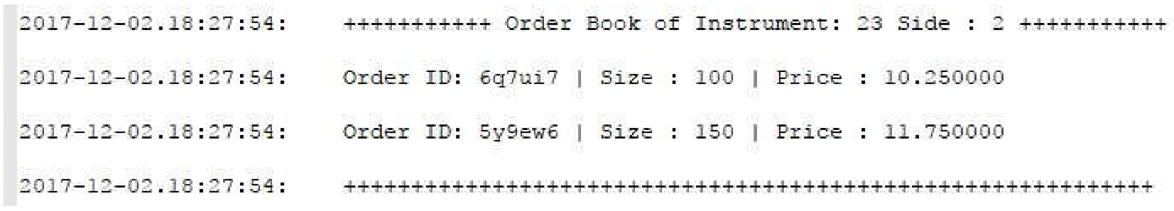
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			10.250000	100	6q7ui7
			11.750000	150	5y9ew6

Table A.6: Putting a sell order with a higher priority



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 6q7ui7 Order Size : 100 Order Price : 10.25
Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.11: Automated test result for test case in Table 5.6



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 6q7ui7 | Size : 100 | Price : 10.250000

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

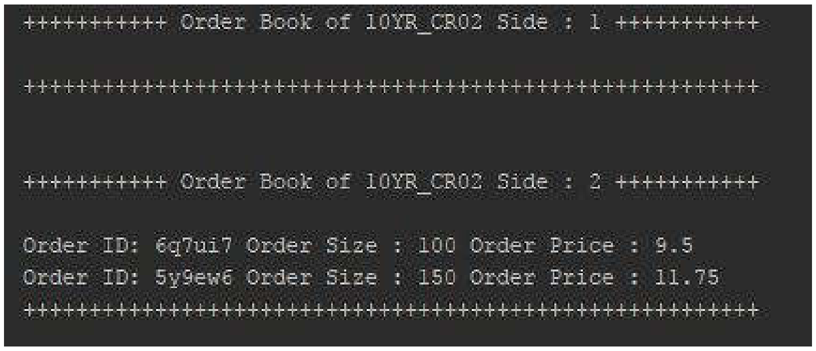
Figure A.12: Actual result extracted from log file for test case in Table 5.6

7. **Amending the previous order (Without losing priority)**

   Now we amend order 6q7ui7 changing its price to 9.5

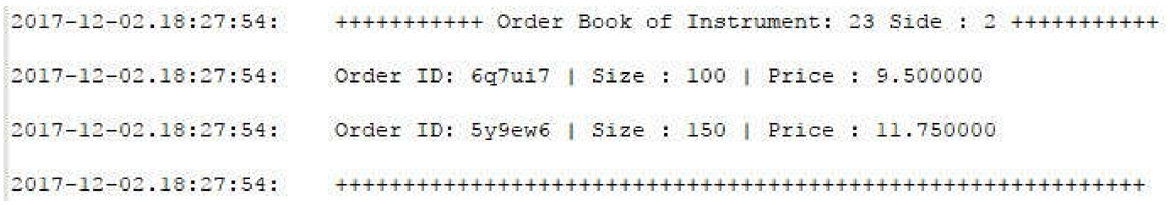
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			9.500000	100	6q7ui7
			11.750000	150	5y9ew6

Table A.7: Order amend without losing priority



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 6q7ui7 Order Size : 100 Order Price : 9.5
Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.13: Automated test result for test case in Table 5.7



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 6q7ui7 | Size : 100 | Price : 9.500000

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

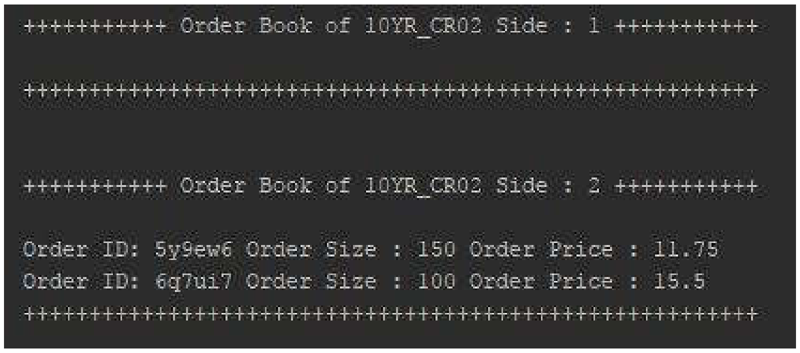
Figure A.14: Actual result extracted from log file for test case in Table 5.7

8. **Amending the previous order (Loosing priority)**

   Now the price of the same order is increased in such a way that its priority is reduced

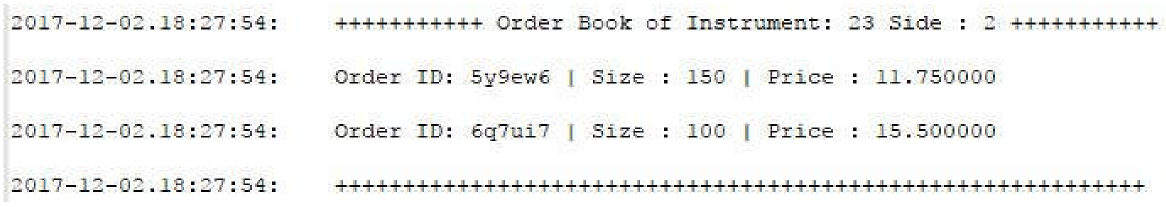
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			11.750000	150	5y9ew6
			15.500000	100	6q7ui7

Table A.8: Order amend while losing priority



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
Order ID: 6q7ui7 Order Size : 100 Order Price : 15.5
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.15: Automated test result for test case in Table 5.8



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    Order ID: 6q7ui7 | Size : 100 | Price : 15.500000

2017-12-02.18:27:54:    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

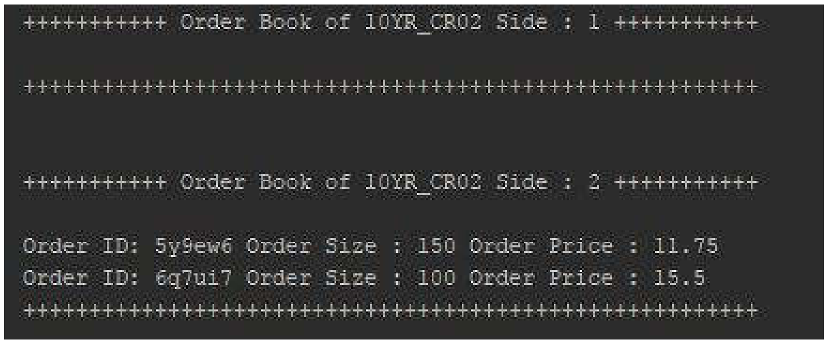
Figure A.16: Actual result extracted from log file for test case in Table 5.8

9. **Put a buy market order**

Now a market order feature is tested and put in size of 100 which priced at 10

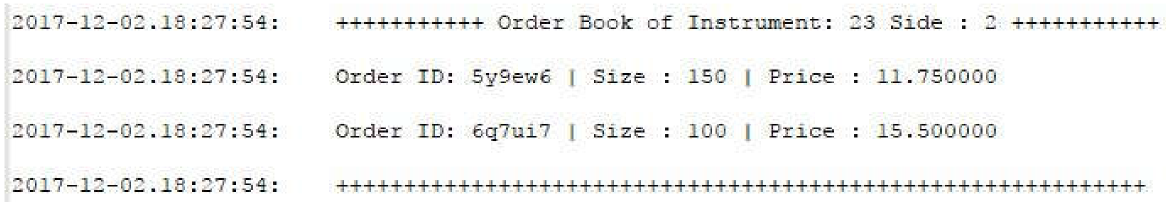
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			11.750000	150	5y9ew6
			15.500000	100	6q7ui7

Table A.9: Market order with expiry



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
Order ID: 6q7ui7 Order Size : 100 Order Price : 15.5
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.17: Automated test result for test case in Table 5.9



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    Order ID: 6q7ui7 | Size : 100 | Price : 15.500000

2017-12-02.18:27:54:    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

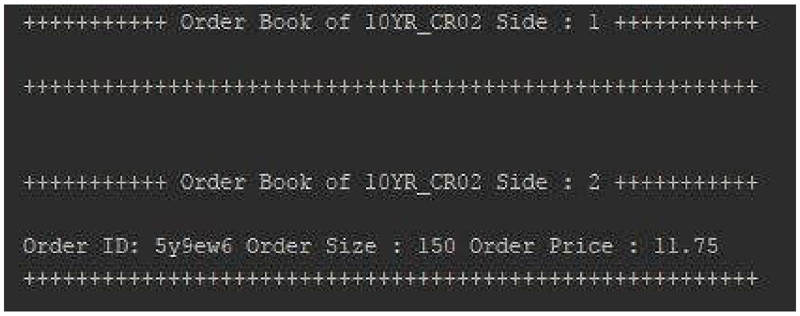
Figure A.18: Actual result extracted from log file for test case in Table 5.9

10. **Order cancellation**

Now order 6q7ui7 is canceled and following should be the outcome

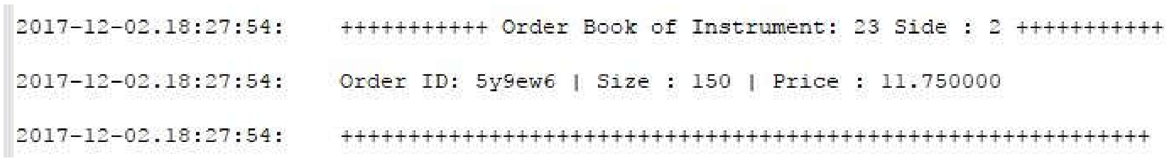
Order Book					
Buy Side			Sell Side		
Order ID	Size	Price	Price	Size	Order ID
			11.750000	150	5y9ew6

Table A.10: Order cancellation



```
+++++++++++ Order Book of 10YR_CR02 Side : 1 +++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++


+++++++++++ Order Book of 10YR_CR02 Side : 2 +++++++++++

Order ID: 5y9ew6 Order Size : 150 Order Price : 11.75
+++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.19: Automated test result for test case in Table 5.10



```
2017-12-02.18:27:54:    +++++++++++ Order Book of Instrument: 23 Side : 2 +++++++++++

2017-12-02.18:27:54:    Order ID: 5y9ew6 | Size : 150 | Price : 11.750000

2017-12-02.18:27:54:    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure A.20: Actual result extracted from log file for test case in Table 5.10