

Formalization, Selection and Detection of Security Patterns

Mohd Suleman



Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

Formalization, Selection and Detection of Security Patterns

Thesis submitted by

Mohd Suleman

[Roll: 710CS2041]

In partial fulfilment of the requirements for the award of the degree
of
Master of Technology
in
Computer Science and Engineering

under the guidance of

Prof. S. K. Rath

NIT Rourkela



Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India



Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

May 25, 2015

Certificate

This is to certify that the work in the thesis entitled *Formalization, Selection and Detection of Security Patterns* by *Mohd Suleman* is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. S. K. Rath
Head of The Department
Department of Computer Science and Engineering
NIT Rourkela

Acknowledgment

I express our sincere and heartfelt gratitude towards our guide Prof. S. K. Rath for his expert guidance and motivation during the course of the project which served as a spur to keep the work on schedule.

We convey our regards to all the other faculty members of Department of Computer Science and Engineering, NIT Rourkela for their valuable guidance and advises at appropriate times. Finally, I would like to thank Ms. Prayasee Pradhan and Mr. Ashish Dwivedi for their help and assistance all through this project.

Mohd Suleman

Abstract

Generally, software requirement analysis and design methodologies based on different UML (Unified Modelling Language) diagrams need to be strengthened by the use of a number of security patterns. Security Patterns provide a way for the software developers to communicate at security level in more comprehensive way. Over the last few years, a number of security patterns has been gradually increased and still increasing. Large number of security patterns has given rise to critical problem of selecting the appropriate security pattern to solve the problem at hand. In this study, an attempt has been made for automated verification of security pattern and an approach is proposed for selection of appropriate security patterns that fulfils security requirements. In order to demonstrate this approach, four security patterns have been selected such as Single Access Point, CheckPoint, Role and Session. A grammar has been developed for the verification of selected security patterns. Goal-Oriented Requirement Language (GRL) has been used for creating the repository of formalized security patterns, this GRL model is used for extracting facts which are then represented as relational instances. Queries have been made to the instances to find appropriate security pattern which fulfils security requirements. This approach clearly identifies the contribution and consequences of a security pattern towards the security related Non Functional Requirements (NFRs). It also checks for the relationships and dependences among the security patterns, which helps in finding the pre-requisite patterns for the selected security patterns. Finally, a method for detection of security patterns using similarity score is presented.

Keywords : Security Patterns, Formalization, Selection, Detection.

Contents

List of Figures	vii
1 Introduction	2
1.1 Introduction	3
1.2 Objective	8
1.3 Organization of the thesis	9
2 Literature Survey	10
2.1 Formalization and Validation of Security Patterns	11
2.2 Selection of Appropriate Security Patterns	13
2.3 Detection of Security Patterns	15
3 Formalization and Validation of Security Patterns	17
3.1 Formalization and Validation of Security Patterns	18
3.1.1 Explanation of Grammar:	18
3.2 Case study	24
3.2.1 Test Cases	28
4 Selection of Appropriate Security Patterns	31
4.1 Modeling of Security Pattern for Building Repository	32
4.1.1 Extraction of Facts from GRL Model	34
4.1.2 Selection of Appropriate Security Pattern that Fulfills Security Requirement	35
5 Detection of Security Patterns	37
5.1 Detection of Security Patterns	38
5.2 Implementation	38
5.2.1 Hybrid Classifier	38
5.2.2 Detection Using Similarity Score	40
6 Conclusion and Future Work	42
6.1 Conclusion	43
6.1.1 Formalization and Validation of Security Patterns	43
6.1.2 Selection of Appropriate Security Patterns	43
6.1.3 Detection of Security Patterns	44
6.2 Future Work	44

List of Figures

1.1	Single Access Point Security Pattern	4
1.2	Check Point Security Pattern	5
1.3	RBAC Security Pattern	5
1.4	Session Security Pattern	6
1.5	Pattern Language for selected patterns	6
3.1	Parser Rules	19
3.2	Lexer Rules	20
3.3	Class Association File	23
3.4	Result : ANTLR Command Prompt	23
3.5	Result : ANTLR Parser Tree 1	23
3.6	Result - ANTLR Parser Tree 2	24
3.7	UML Class Diagram for Case Study	25
3.8	Extended UML class Diagram for Case Study	26
3.9	Result - ANTLR Command Prompt	28
3.10	Result - ANTLR Parser Tree	28
3.11	Test-Case 1	29
3.12	Test-Case 2	29
3.13	Test-Case 3	30
4.1	GRL intentional elements	32
4.2	GRL model of Check Point Security Pattern	33
4.3	GRL model of Role-Based Access Control Security Pattern	33
5.1	jRefactory - Composite Pattern Detection Result	39
5.2	jRefactory - Adapter Pattern Detection Result	39
5.3	Assosiation Matrix For Single Access Matrix	41

List Of Abbreviation

UML	Unified Modeling Language
GRL	Goal-Oriented Requirement Language
NFR	Non Functional Requirement
SAP	Single Access Point
RBAC	Role Based Access Control
ANTLR	ANother Tool for Language Recognition
OMT	Object Modeling Technique
XML	eXtensible Markup Language
MARPLE	Metrics and Architecture Reconstruction Plug-in for Eclipse
DTNB	Decision Tree Naive Bayes

Chapter 1

Introduction

1.1 Introduction

In the past two decades, a good number of software patterns have been proposed by researchers [1] [2] [3] [4] [5] [6] [7]. Many design pattern tools have also been developed for detecting patterns in instantiating of design patterns [8]. Gamma et al. [1] have proposed the concept of software design patterns which consist of the standard templates for twenty three design patterns. Later, other software design patterns used these templates as a base and further extended these templates for their application area.

The security requirements of a system depend on the environment in which system is developed. In the present day scenario, the aspect of software security is different from end-to-end security requirements of an application. Security principles say that by eliminating security risks at the functional and developmental level, security business objects, data across logical tiers, and security communications can be improved. Also the protection of the application from unauthorized internal and external threats and vulnerabilities need to be ascertained. For the first time security patterns have been proposed by Yoder and Barcalow [9]. They have proposed seven patterns which are applied in security development issues. Later, a good number of other categories of security patterns have been introduced [5] [6] [10]. In order to demonstrate our approach four security patterns such as Single Access Point, Check Point, Session, and Role have been taken into consideration. These security patterns are described as follows:

The single Access Point limits extraneous access to a single channel and facilitates control which may be used in any self-contained system communicating with others. Single Access Point security pattern provides a scheme for static design of a system. Many systems can't be protected against outside attacks due to numerous access points. Hence, the main objective of Single Access Point is to define one single interface for all external entities to communicate with system. The Single Access Point is used at the network-level as well as the application-level. The UML class diagram for Single Access Point is shown in Figure 1.1. Three participants for Single Access Point (SAP) are 'Internal Entity', 'Single Access Point', and 'External Entity'. External Entities are the actors which are outside the system and should be authenticated before they can communicate with the system. They communicate with the system through Single Access Point. Internal Entities are the components present inside the system which is accessible to the

external entities only when authenticated by Single Access Point available to the external entities.

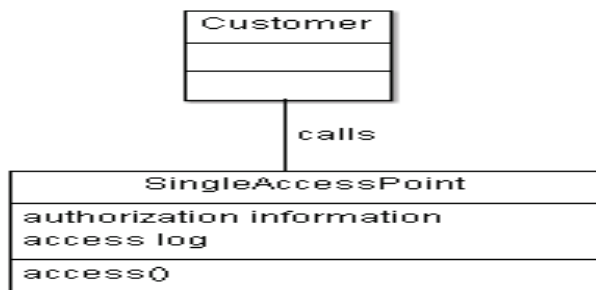


FIGURE 1.1: Single Access Point Security Pattern

The Single Access Point (SAP) class interacts with any class that needs to communicate with external entities. All the security policies that are to be enforced, must be sent to Check Point class before requests are forwarded to their intended recipients. Check Point performs a check for security policies that are to be applied on system and penalizes the user for violation of security policies.

A Single Access Point is predesigned to be combined with Check Point. Class diagram for Check Point is shown in Figure 1.2. Check Point class acts as an Internal Entity to the Single Access Point class and checks for information flow through this class. Three participants for Check Point security pattern are Check Point, Countermeasure and Security Policy. The Check Point class implements a method to check information according to the current security policy and triggers action to protect system against attackers. Countermeasure class provides actions to be triggered in order to react against access violation. Security Policy class implements rules that are applied to determine whether an access or condition is allowed or not.

Check Point grants access to the system for authorized users. The user must be validated for the authorization areas of the system according to the role that particular user plays. Validation and verification for user's privileges are incorporated through Role Based Access Control (RBAC) security pattern. The class diagram for RBAC is shown in Figure 1.3. RBAC security pattern aims for better maintainability of the privileges in the system. RBAC security pattern is realized by constructing a User-Role-Privilege relationship. The different participants taking action in RBAC security pattern are Privilege and Role. Privilege class defines the resources which are accessible to subjects that has been afforded to

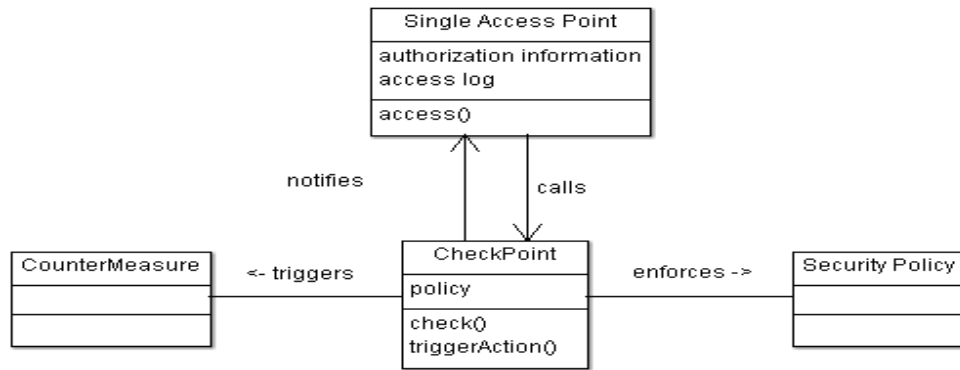


FIGURE 1.2: Check Point Security Pattern

this privilege. Role class holds a set of privileges those are related to that specific role object and furnishes information about the users and its privileges.

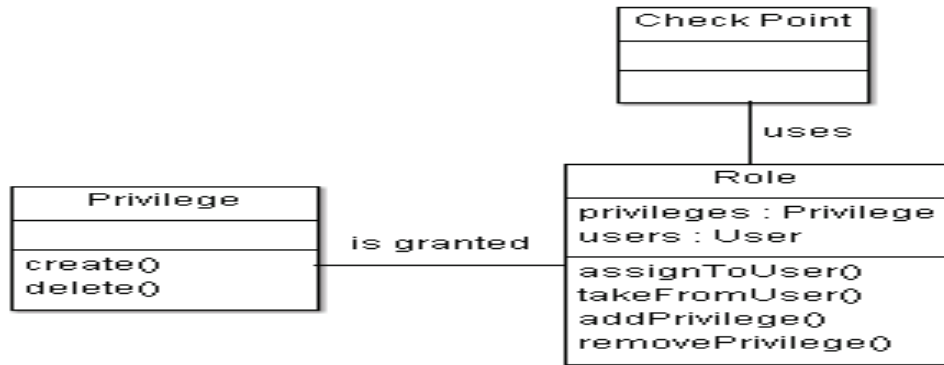


FIGURE 1.3: RBAC Security Pattern

Session security pattern deals with creation of object which keeps global information about a user. This may be used to facilitate accountability and to enforce privilege violations. Session objects keep security relevant information like roles, privileges or authentication data. Session object may be created after the user logs into the system which is to be done at the Check Point of system. The UML class diagram for Session security pattern is shown in Figure 1.4. The participants of this security pattern are Session and the System components which uses Session. Session class stores information which are provided by Session. The stored information are initialized on creation. System components, which use Session, know the instance of the Session object they use, and call methods of session to retrieve information.

According to Yoder and Barcalow [9], secure system should maintain a proper associativity among different security patterns. In this study, four security patterns

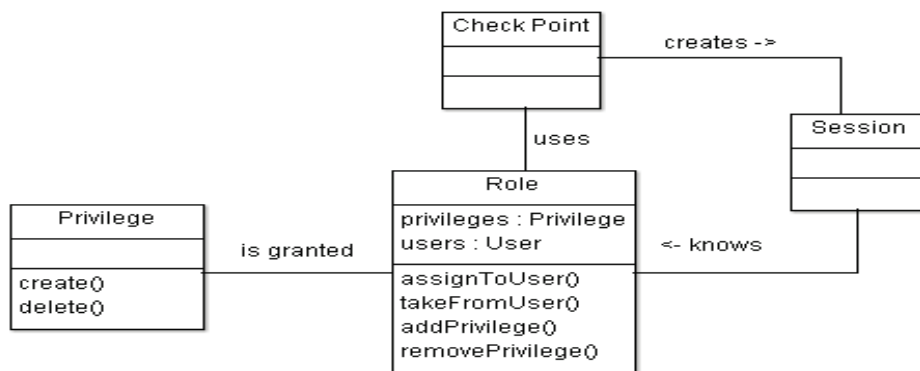


FIGURE 1.4: Session Security Pattern

such as *Single Access Point*, *Check Point*, *Session*, and *Role* have been taken into consideration. One of the important measures of security is *Single Access Point* which limits the entry to the system only through a single entry point. *Single Access Point* provides user identification related information to *Check Point* for authentication and authorization of the user. When user identification has been verified, *Session* is created for carrying the global variables which contain user's identification and role. The authorization area for system visualization and modification is provided through role-privilege relationship. Figure 1.5 represents the flow of associativity among the aforesaid patterns in order to provide a secure system.

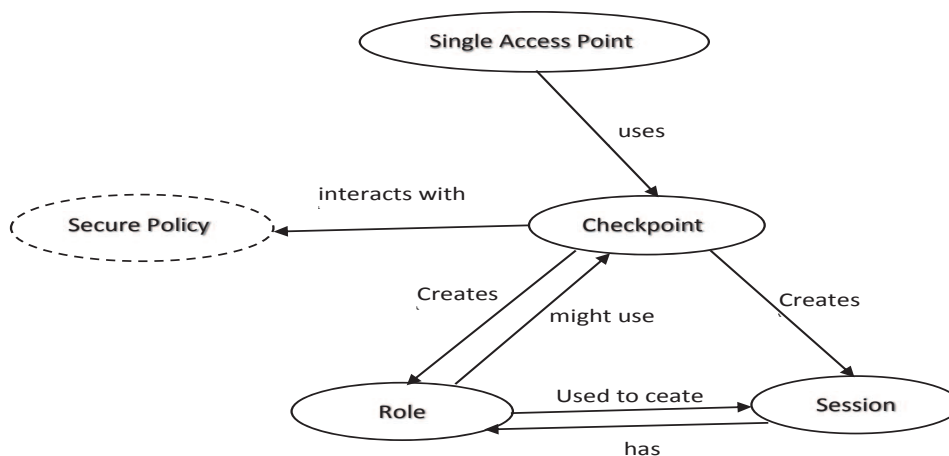


FIGURE 1.5: Pattern Language for selected patterns

Design of an application system at present is supposed to be based on different UML diagrams. UML class diagram shows the structural aspects of the classes, but it is unable to express some other behavioral aspects. Hence, the extension of UML class diagram to visualize the design pattern methodology has been proposed

in [11]. A gradual evolution has been observed for representation of design patterns in UML class diagram, including Venn diagram style notation, Dotted-Bounding Pattern Annotation, and Tagged Value Notation. Tagged Value notation defines the pattern-role behavior of the model elements such as classes, attributes, and operations. These notation can also be used for representation of security patterns. Dong et al. [12] have presented a UML profile that defines new stereotypes, tagged values, and constraints for tracing design patterns in UML diagrams. In this study, these notations [12] are used for improving analysis of security problem.

The role of formal methods in the area of design patterns is helpful to enhance the understandability of their semantics. It can help to analyse the composition of design patterns. In the verification and validation scenario, the design must be checked for correct use of the design patterns. There are many advantages of formal notations to improve design decisions. Formal notations provide new methods to prove the correctness of the proposed design, to automatically check the syntax and semantics of the design decisions, and to generate test cases [13]. The verification and validation of any requirement are being carried out by the use of formal languages which are based on grammar and have certain production rules. Large numbers of formal specification languages are available in the literature. Each language has its own syntax and semantics. The limitation of formal language is that a single formal specification language can't be applied universally for all types of problems. For example, Taibi and Ngo [14] proposed a balanced approach between structural and behavioral aspect, but the approach does not provide any information like the role of the participating model elements such as classes, attributes, and methods in a particular design pattern.

Different approaches have been proposed to deal with the problem of selecting appropriate security patterns. Prevailing approaches emphasize on formalization of security pattern in order to create the repository. Formalization of security pattern provide the clear description of the structure of security patterns. Structure of security pattern helps the software developer in understanding, how the security pattern can be applied but it does not provide any information on when a security pattern should be applied for a given security related problem.

In this study, formal specification in the form of grammar is proposed for verification of security patterns. The extended UML class diagram is verified by ANTLR using proposed grammar. All the security goals as intended by selected

security pattern are preserved after the verification of extended UML class diagram. A centralized repository for storing security pattern is created. Queries has been made to the repository in order to find the most appropriate design pattern that fulfills Non Functional Requirements. A approach using similarity score has been proposed in order to detect the security patterns from the provided source code of a system.

1.2 Objective

Security is one of several non-functional requirement that software developers have to consider during the software development lifecycle [18]. Recently, applications have become increasingly large and complex and developers might now have the kind of security expertise that the design of security system requires. Security Patterns bridge the knowledge gap between software developers in different domains, and especially between software security specialists and software designers. A security pattern [15] is a well-understood and well-formed solution to a recurring software design security problem. So, security patterns encapsulate the knowledge accumulated by security experts in order to help software developers to develop a secure software system

As the requirement for security of today's systems is continuously increasing, the number of security is also gradually increasing. Recently many number of security patterns has been proposed by researchers. Rising's Pattern directory [16] has listed more than twelve hundred patterns. In the past ten years since the publication of the Rising pattern directory, many new patterns book related to security pattern have been published. In 2007 list of two thousand two hundred and forty one patterns focusing solely software was given by Henninger and Correa [17]. Increase in number of security can be considered as beneficial for the development of secure system but it gave given rise to new critical problem, the problem of selection the most appropriate pattern from the pool containing many security pattern. The problem of selecting pattern was first highlighted by Gamma (also known as GoF), he said that it can be hard to find most appropriate pattern for the problem at hand from the pool containing more than twenty patterns [17].

In this study, after the formalization of security pattern using grammar, GRL model has been used to represent the contribution that a security pattern make

on the security related Non Functional Requirements (NFRs), it identifies which NFR will be build and which NFR will be hurt by the use of particular security pattern. GRL modelling also helps in visualizing the different relationship among the patterns such as, weather a pattern can co-exist with other patterns and what are the prerequisite patterns for the particular pattern. Therefore, GRL model is used for creating a repository of security pattern. Facts are extracted from the GRL model which is then represented in the form of Instances. Queries are made to these instances for finding the appropriate security pattern that fulfils security requirement.

1.3 Organization of the thesis

The thesis is organized as follows.

In chapter 2, the literature survey is provided, which focuses on different works that has already been undertaken in the field of formalization, validation, selection and detection of security patterns .

In chapter 3, a methodology for the formal verification of selected security patterns has been discussed. Developed grammar which satisfies the proposed pattern language has been explained.

In chapter 4, a methodology for the selection of appropriate security patterns has been proposed. Implementation details of the approach is also shown in the same chapter.

In chapter 5, a methodology for detection of security patterns using similarity score have been propose. Open source projects have been considered to find correctness of the proposed method.

In chapter 6 presents the concluding remarks with a focus on future research directions that could be undertaken.

Chapter 2

Literature Survey

2.1 Formalization and Validation of Security Patterns

The very first notation used for identification of design patterns in UML diagram was *Venn Diagram style Pattern Annotation* [19]. In this method, the model elements participating under the same pattern are clustered together. The concept is well accepted for small system, but clustering of elements in a larger system was not possible due to the lack of simplicity and overlapping of clusters. This method simply shades the cluster with a color in order to make it distinguishable from other ones, but still it was not widely accepted for large system.

In order to prevent the shortcoming of shading problem of *Venn Diagram style Pattern Annotation*, the *Dotted Bounding Pattern Annotation* was developed by Dong [20]. But still the notations were imprecise to decide the exact role of the model elements which they play under the particular design pattern.

Berner et al. have proposed a notation based on UML stereotypes called as *restrictive stereotype* [21]. The method defined the design pattern and role of the model elements participating in a system. But, the stereotype notation was difficult to handle in terms of expensiveness of designing, using and maintaining the notation. Also, their approach was not clear about how to extend UML stereotype notation to represent the compositions of design patterns.

Dong have proposed a new notation to represent explicitly the roles of each class, operation, and attribute in a pattern, which is based on an extension to UML [11]. The extension was defined mainly by applying the UML built-in extensibility mechanisms. The new notation was called as *Tagged Pattern Annotation*. This method also fulfilled the drawbacks of the *Stereotype Annotation Pattern* by allowing the representation of composition of design patterns.

T.Taibi and D.C.L. Neo [14] proposed a formal notation known as, BPSL (Balanced Pattern Specification Language). The main aim of this language was to combine two subsets of Logic, one from First Order Logic (FOL) and other from Temporal Logic of Actions (TLA). According to authors, BPSL has carefully chosen the subsets of FOL and TLA to be used in order to be simple for users and yet described design patterns accurately. The ultimate purpose of BPSL is to help users to understand patterns to know exactly when and how to use them.

Dong et al. [22] proposed an approach to automate the verification of the compositions of security patterns by model checking. They formally described the behavioral aspect of security patterns in CCS (Calculus of Communicating Systems) through their sequence diagram. They also proved the faithfulness of the transformation from a sequence diagram to its CCS representation. In their research, they used two case studies to demonstrate their approach and shown its capability to detect composition errors. Dwivedi and Rath [23] formalized a complex architectural style i.e., C2 (Component and Connector) using formal modeling language Alloy. They have considered cruise control system as a case study.

Bayley and Zhu [24] proposed a meta modeling approach toward formalization of design patterns. This approach enables formal reasoning about patterns and their composition, transformation, and facilitates automatic tool support for applying patterns at the design stage. For the case study, authors have formally specified all 23 Gamma's design patterns. They claimed that the class diagram of facade pattern given by GoF [1] is not even well formed and cannot be taken at facevalue in terms of either the number of classes or their interconnections. Dwivedi and Rath [25] have formalized an architectural style C2 using formal modeling languages Alloy and Promela. For the model checking of these formal notations, automated verifiers such as Alloy Analyzer and SPIN are used.

Dey and Bhattacharya [26] have proposed a formal specification language FSDP (Formal Specification of Design Pattern) to formally specify design patterns from UML class diagram. They have used ANTLR (ANother Tool for Language Recognition) for verification of their developed grammar. They developed a tool from FSDP grammar to formally automate pattern design techniques, to create, store, and retrieve UML class diagrams within design patterns. The proposed grammar is only able to verify the notation [11] for representing design patterns in extended UML class diagram. Grammar verifies textual format of extended UML class diagram but it does not check associativity between the different design patterns and it also fails to check correct placement of roles for design patterns.

2.2 Selection of Appropriate Security Patterns

To select a precise implementation for a software design a cognitive model was developed by Hinojosa [27]. This cognitive model deals with the behavioural design patterns from Gamma et al. [17] because of the implementation strategies implied by those patterns. A reasoning engine based on Prolog language was developed, which consumed the set of features that were mapped to the patterns. Real world implementation decision data was used in order to deduce which feature will help in guiding engineers to a specific implementation. Thus, providing a set of relevantly appropriate features for each behavioural design pattern. This approach provides a cognitive model for human reasoning for selecting appropriate security pattern. However, it is observed that this approach is not applicable in various domain and this approach is not sufficiently scalable, because all patterns are required to be transformed manually into sets of predicates. Decisions for selection of appropriate patterns should also be manually processed.

Patterns-Box tool for assisting software developers in designing a software architecture was developed by Albin Amiot et al. [28]. All the pattern were modelled with the help of Formal Pattern Description Language (PDL), in order to create the repository. For the selecting appropriate pattern, formal model for current application context has been used. Patterns-Box tool also provides the HTML interface to navigate between the patterns. However, it is observed that this approach did not emphasize on relationship and dependencies among the patterns

Design pattern recommendation system which satisfies the contextual requirements such as security, privacy was proposed by Pearson and Shen [29]. Rules based engine was developed which takes context requirement of the required design as an input. For selecting appropriate pattern, engine triggers decision about pattern based on the input. This approach is targeted to help non expert software architects and developers. Patterns are connected with rules, which make them independent of the representation format. Therefore patterns and rules much be created based on the industry practices in each domain. This system is an expert system which takes selection decision based on the knowledge represented in form of rules.

An approach for selecting appropriate design patterns that fulfills the non functional requirements of the architectural design was presented by Wang et al. [30].

A prioritized list of suitable patterns for a specific set of requirements is retrieved with the help of Non Functional Requirements (NFR) framework. AND or OR relationships for each pattern is identified hierarchically. These relationships are then used for analysis of the traceability from the software design components to the software architecture components. Appropriate applicability of design pattern is obtained by the use the NFR framework.

Goal-Oriented Requirement Language (GRL) that formalizes the relation between the patterns and forces of patterns was proposed by Mussbacher et al. [31]. The formalization and clear representation of forces enables trade-off analysis of forces during the selection of appropriate security patterns. Formalization of security pattern with the help of GRL graphs helps in capturing the pattern forces, and it also helps in assessing the qualitative influence on numerous solutions to required functional goal. GRL model also helps in identifying the relationships and dependencies among the patterns. Therefore this approach supports selections of combinations of security patterns. At the moment author did not provide any tool based on this selection approach.

Weiss and Mouratidis [32] have extended the work of Mussbacher et al. [31]. Formalization was done with the help of same GRL model. Their approach appends few more steps to the approach followed by the Mussbacher et al. [31]. Facts were extracted from the GRL model and were stored in Prolog for reasoning. For selecting appropriate security pattern user makes query to the Prolog engine, which returns the list of security patterns which fulfils the requirements specified in the query. Subsequently, this approach also check for the relationship and dependencies among the pattern and return the list of prerequisite patterns. However, Prolog is client side language updating security pattern repository will be tedious tasks and it is observed that it is difficult to provide centrally managed pool of security pattern using this approach. This work is the extension of the work done by Weiss and Mouratidis[32]. In this study approach repository of security pattern is stored centrally in server and an interface is provided to make query to the repository in order to get the list of appropriate security pattern.

2.3 Detection of Security Patterns

A system has been developed by Prechelt and Kramer [33] to detect a number of design patterns present in C++ source code. They have built OMT class diagrams, which incorporates design patterns, to provide Prolog rules. Hence, new Prolog rules were proposed to detect the design pattern instances.

Wendehals [34] has proposed a method to detect the design pattern instances present in a system by using the combination of static and dynamic analysis. UML class diagrams are used to retrieve static information and dynamic information was extracted from Collaboration or Sequence Diagrams.

Heuzeroth et al. [35] have proposed a methodology to perform static analysis to obtain pattern candidates, and dynamic analysis of pattern instances was carried out on the previously obtained candidate sets from static analysis. Since, static and dynamic analysis are dependent on the formation of design patterns, distinct algorithms are needed to perform the static and dynamic analysis for each design pattern. Hence, it is difficult to develop an automated design pattern detection methodology.

A technique has been proposed by Antoniol et al. to detect structural patterns in a software system [36]. The approach uses metrics to identify probable pattern candidates, and distance measures are considered for roles in the patterns in the next stage. In the final stage, delegation constraints are generated. The three stage approach aims at reducing search space. Final pattern instances are identified based on structural information. Therefore, this approach has got very low precision for pattern matching.

Balanyi et al. have proposed an approach to extract the abstract semantic graph as well as DPML (Design Pattern Markup Language) by the help of Columbus reverse engineering framework [37] [38]. The proposed approach matches roles present in the DPML files, and the exploration space is reduced by filtering based on structural information. This approach performs exact matching, hence it can't identify modified pattern versions.

Tsantails et. al. have proposed a methodology to detect a design pattern based on similarity scoring between graph vertices which is capable of recognizing patterns that are modified from their standard representations [39]. Instead of relying on pattern-specific heuristic, the approach reduces the search space by

taking the fact into consideration that pattern resides in one or more inheritance hierarchies.

Chapter 3

Formalization and Validation of Security Patterns

3.1 Formalization and Validation of Security Patterns

In this study, a formal specification in the form of grammar is proposed for verification and validation of security patterns. Proposed grammar is based on the pattern language which is shown in Figure 1.5. Four basic security patterns taken into consideration are *Single Access Point*, *Check Point*, *Session*, and *RBAC*. The system may contain other security patterns, but presence of security patterns in a particular association is of very much significance for secure system. Any language which is accepted by the proposed grammar, may be said to preserve security aspects. The extended UML diagram is verified with the help of proposed grammar.

User is able to define security patterns as well as the role to define the behavioral characteristics of model elements such as class, attribute, and method. The main aim of this study is to verify the associativity of Pattern-Class containing security patterns. A system having pattern language of security patterns as shown in Figure 1.5, is abide to preserve security. Proposed grammar can verify extended UML class diagram against the pattern language shown in Figure 1.5. The grammar is developed according to the specification of ANTLR. The parser rule and lexer rules for the proposed grammar are given in Figure 3.1 and Figure 3.2 respectively.

3.1.1 Explanation of Grammar:

The class associativity file contains the association of classes, i.e. classes associated to a class either by association, by dependency, by generalization or by aggregation. The first rule verifies the validity of the class. Simultaneously, it checks for the syntax for declaring security patterns associated with the class. The class name must be a valid declaration, and must be having the pattern *SingleAccessPoint* and role *ExternalEntity*.

```
classDecl : className sapExternal;
```

```
sapExternal : LEFTBRACE ( patternSAP (patternInstance)? SLASH role-
SAPEXternal (COMMA)?+ ((COMMA)? patternName (patternInstance)? SLASH
```

```

classDecl : className sapExternal;
sapExternal : LEFTBRACE ( patternSAP (patternInstance)? SLASH roleSAPEXternal (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE classSAPSINGLETON;
classSAPSINGLETON : LEFTBRACKET className sapSingleton;
sapSingleton : LEFTBRACE ( patternSAP (patternInstance)? SLASH roleSAPSINGLETON (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? classCheckPoint;
classCheckPoint : LEFTBRACKET className checkPoint;
checkPoint : LEFTBRACE ( patternCheckPoint (patternInstance)? SLASH roleCPCHECKPOINT (COMMA)?+ ((COMMA)?
patternSAP (patternInstance)? SLASH roleSAPINTERNAL)+ ((COMMA)? patternName (patternInstance)? SLASH role)*
RIGHTBRACE (RIGHTBRACKET)? classSecurityPolicy;
classSecurityPolicy : LEFTBRACKET className securityPolicy;
securityPolicy : LEFTBRACE ( patternCheckPoint (patternInstance)? SLASH roleSP (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? classCounterMeasure;
classCounterMeasure : className counterMeasure;
counterMeasure : LEFTBRACE ( patternCheckPoint (patternInstance)? SLASH roleCM (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? classSession;
classSession : className session;
session : LEFTBRACE ( patternSession (patternInstance)? SLASH roleSession (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? classRole ;
classRole : className rbac;
rbac : LEFTBRACE ( patternRBAC (patternInstance)? SLASH roleROLE (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? classPrivilege ;
classPrivilege : LEFTBRACKET className rolePrg;
rolePrg : LEFTBRACE ( patternRBAC (patternInstance)? SLASH rolePrg (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? finalSessionClass ;
finalSessionClass : className finalSession;
finalSession : LEFTBRACE ( patternSession (patternInstance)? SLASH roleSession (COMMA)?+ ((COMMA)? patternName
(patternInstance)? SLASH role)* RIGHTBRACE (RIGHTBRACKET)? endclass;
endclass : (RIGHTBRACKET)+ ((LEFTBRACKET)? (className patternDecl)+ (RIGHTBRACKET)* )* ;
patternDecl : LEFTBRACE patternName (patternInstance)? SLASH role (COMMA patternName (patternInstance)? SLASH role)*
RIGHTBRACE;
patternInstance : LEFTBRACKET instanceNo RIGHTBRACKET;
patternSAP : SINGLEACCESSPOINT;
roleSAPEXTERNAL : EXTENTIVITY;
roleSAPINTERNAL : INTENTIVITY;
roleSAPSINGLETON : SAPSINGLETON;
patternCheckPoint : CP;
roleCPCHECKPOINT : CP;
roleSP : SP;
roleCM : CM;
patternSession : SS;
roleSession : SS;
patternRBAC : 'RBAC';
roleROLE : 'Role';
rolePrg : 'Privilege';
entitynumber : INTEGER;
instanceNo : INTEGER;
className : STRING;
patternName : STRING;
role : STRING;

```

FIGURE 3.1: Parser Rules

```

EXTENTITY : 'ExternalEntity';
SAPSINGLETON : 'Singleton';
INTENTITY : 'InternalEntity';
CP : 'CheckPoint';
SP : 'SecurityPolicy';
CM : 'CounterMeasure';
SS : 'Session';
WS : [\t\r\n]+ ->skip;
COMMA : ',';
SLASH : '/';
LEFTBRACE : '{';
RIGHTBRACE : '}';
LEFTBRACKET : '[';
RIGHTBRACKET : ']';
SINGLEACCESSPOINT : 'SAP';
STRING : (CHAR)+;
INTEGER : (DIGIT)+;
CHAR : ('a'..'z'|'A'..'Z')('a'..'z'|'A'..'Z' | '_' | ':' | '-' | '0'..'9')*;
DIGIT : ('0'..'9');

```

FIGURE 3.2: Lexer Rules

role)* RIGHTBRACE classSAPSingleton;

Above rule indicates class declaration must be according to the syntax
ClassName{*PatternName*[*Instance*]/*Role*}

The parser rule for *patternSAP* is as follows:

patternSAP : SINGLEACCESSPOINT;

The lexer rule for *SINGLEACCESSPOINT* is as follows:

SINGLEACCESSPOINT : 'SAP';

(*patternInstance*)? says that the rule may or may not contain *patternInstance*, which means it is optional. Pattern instance is used when a same security pattern is present more than once in a UML class diagrams. Basically, *patternInstance* helps to distinguish between different roles of same security patterns.

patternInstance : LEFTBRACKET instanceNo RIGHTBRACKET;

The above rule can be explained with the help of an example, *[1], [2], or [9] etc* is accepted by the rule *patternInstance*. Here, '[1]' represents the first presence of the security pattern, '[2]' represents the second presence of same security pattern and so on.

The parser rule for *roleSAPExternal* is as follows:
roleSAPExternal : EXTENTITY;

The lexer rule for EXTENTITY is as follows:
 EXTENTITY : 'ExternalEntity';

((COMMA)? *patternName* (*patternInstance*)? SLASH *role*)* :

The above grammar says that the rule "*((COMMA)? patternName (patternInstance)? SLASH role)*" may occur multiple number of times or it may not occur. Which signifies that each class must play at least one role of any security pattern. If a class does not contain any security pattern then it should be in the form *classnameNull/Null*.

The rule *sapExternal* enforces the presence of the pattern *SingleAccessPoint* having the role *ExternalEntity*. The class may play any other role under any other security pattern, but presence of *SingleAccessPoint* and the role *ExternalEntity* is must. The presence of stereotype defined as *SAP/ExternalEntity* is verified by this rule.

The rule *sapExternal* leads to the verification of the security pattern *SAP/Singleton* by using another rule *classSAPSingleton*. The reason to choose *Singleton* as the role is to provide single access point to the user. Definition of rule *classSAPSingleton* is given below:

classSAPSingleton : LEFTBRACKET *className* *sapSingleton*;
sapSingleton : LEFTBRACE (*patternSAP* (*patternInstance*)? SLASH *roleSAPSingleton* (COMMA)?)⁺ ((COMMA)? *patternName* (*patternInstance*)? SLASH

```
role)* RIGHTBRACE (RIGHTBRACKET)? classCheckPoint;
```

The parser rule for *roleSAPSingleton* is as follows:

```
roleSAPSingleton : SAPSINGLETON;
```

The lexer rule for *SAPSINGLETON* is as follows:

```
SAPSINGLETON : 'Singleton';
```

The class declaration may or may not contain other pattern-role pair, but it must contain at least one matching pattern of *SAP/Singleton*. This constraint is verified by the rule *sapSingleton*.

Association of external entity and singleton under the pattern Single Access Point leads to the verification of class containing pattern *SAP/Singleton* associated with another class containing patterns *SAP/InternalEntity* and *CheckPoint/CheckPoint*. This constraint is verified by using the rule *classCheckPoint*. Definition of rule *classCheckPoint* is given below:

```
classCheckPoint : LEFTBRACKET className checkPoint;
```

```
checkPoint : LEFTBRACE ( patternCheckPoint (patternInstance)? SLASH
roleCPCheckPoint (COMMA)?+ ((COMMA)? patternSAP (patternInstance)?
SLASH roleSAPInternal)+ ((COMMA)? patternName (patternInstance)? SLASH
role)* RIGHTBRACE (RIGHTBRACKET)? classSecurityPolicy;
```

The sample of association text file generated from the extended UML diagrams is shown in Figure 3.3. The result generated by using the grammar proposed by Dey and Shouvik [26] is shown in Figure 3.4, Figure 3.5, and Figure 3.6. Figure 3.4 shows the command prompt output. Figure 3.5, and Figure 3.6 shows the parse tree generated by ANTLR tool.

```

Customer { SAP/ExternalEntity}
[Login { SAP/Singleton}
 [ Verification{CheckPoint/CheckPoint, SAP/InternalEntity}
  [ SecurePolicies{CheckPoint/SecurityPolicy}
   Penalties{CheckPoint/CounterMeasure}
   Sessions{Session/Session}
   ManagingRoles{RABC/Role}
   [ UserPrivileges{RABC/Privilege}
    Sessions{Session/Session}
  ]
 ]
 ]
 ]
 ]
 ]
 ]

```

FIGURE 3.3: Class Association File

```

C:\Windows\system32\cmd.exe - grun securepattern validPattern -tree -gui FSDP1.txt
Connection)) (dependency (className DB2ConnectionPool) (className DB2Connection
)) (dependency (className OracleConnectionPool) (className ConnectionPool)) (depe
ndency (className DB2ConnectionPool) (className ConnectionPool)) (dependency (cl
assName OracleConnection) (className Connection)) (dependency (className DB2Conn
ection) (className Connection)))
C:\antlr\securepattern>grun securepattern validPattern -tree -gui FSDP1.txt
C:\antlr\securepattern>java org.antlr.v4.runtime.misc.TestRig securepattern vali
dPattern -tree -gui FSDP1.txt
validPattern (structure (validClass (classDecl (className OracleConnectionPool)
) (patternDecl ( (annotation AbstractFactory / (role1 createProduct)) )))) (st
ructure (validMethod (methodName createConnection) (parameterDecl ( )) (patternD
ecl ( (annotation AbstractFactory / (role1 createProduct)) )))) (structure (vali
dClass (classDecl (className ConnectionPool) (patternDecl ( (annotation Abstract
Factory / (role1 AbstractFactory)) )))) (structure (validMethod (methodName cre
ateConnection) (parameterDecl ( )) (patternDecl ( (annotation AbstractFactory /
(role1 createProduct)) , (annotation Singleton (patternInstance I (instanceNo 1)
) / (role2 Instance)) )))) (structure (validClass (classDecl (className DB2Con
nectionPool) (patternDecl ( (annotation AbstractFactory / (role1 createProduct)
)) , (annotation Singleton (patternInstance I (instanceNo 2) ) / (role2 Single
ton)) )))) (structure (validMethod (methodName createConnection)) (parameterDecl
( )) (patternDecl ( (annotation AbstractFactory / (role1 createProduct)) , (ann
otation Singleton (patternInstance I (instanceNo 2) ) / (role2 Instance)) ))))
(structure (validClass (classDecl (className OracleConnection) (patternDecl ( (a
notation AbstractFactory / (role1 createProduct)) )))) (structure (validClass
(className DB2Connection) (patternDecl ( (annotation AbstractFactory
/ (role1 createProduct)) )))) (structure (validClass (classDecl (className
Connection) (patternDecl ( (annotation AbstractFactory / (role1 AbstractProduct)
) )))) (behavior (dependency (className OracleConnectionPool) (className Oracle
Connection)) (dependency (className DB2ConnectionPool) (className DB2Connection)
) (dependency (className OracleConnectionPool) (className ConnectionPool)) (depe
ndency (className DB2ConnectionPool) (className ConnectionPool)) (dependency (c
lassName OracleConnection) (className Connection)) (dependency (className DB2Conn
ection) (className Connection)))))

```

FIGURE 3.4: Result : ANTLR Command Prompt

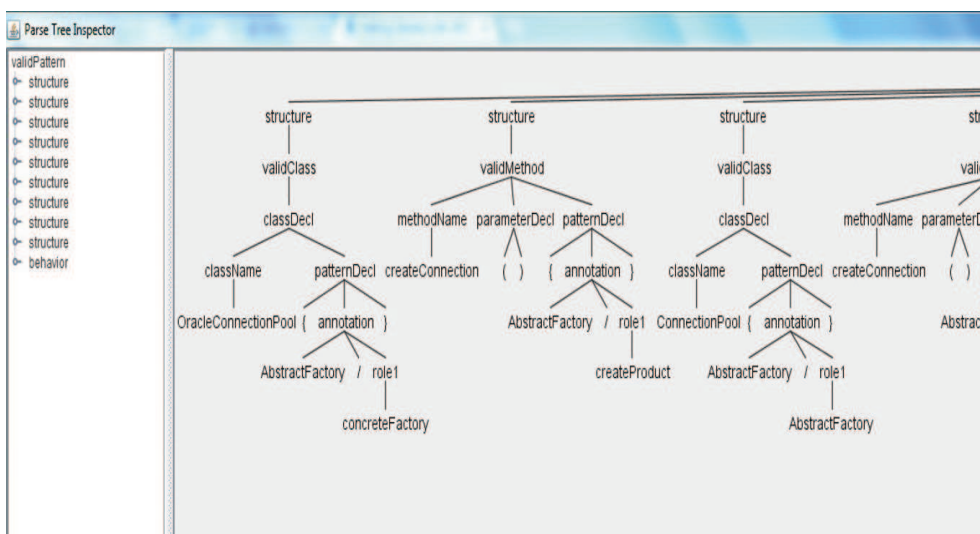


FIGURE 3.5: Result : ANTLR Parser Tree 1

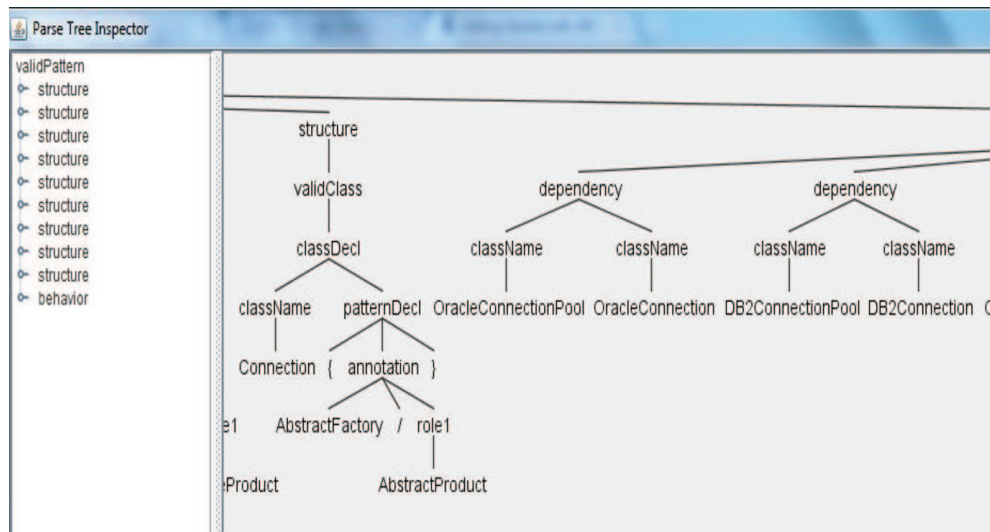


FIGURE 3.6: Result - ANTLR Parser Tree 2

3.2 Case study

In order to demonstrate this approach, case study of online banking system have been considered. Nowadays, customers need more personal security, more advocacy and, more control in their banking relationships. The major challenge with different banks is that they are looking to gain the flexibility, shared services, easy to use and align business to technology. The solution of above challenges can be found with the help of security patterns. In online banking system, customer performs online financial transactions, which requires more security provision. The UML class diagram and extended UML class diagram for the online banking system are presented in Figure 3.7 and Figure 3.8 respectively.

Figure 3.7 shows the class diagram of Online Banking System for incorporating security features. This diagram contains eleven classes such as *Customer*, *Login*, *Verification*, *SecurePolicies*, *Penalties*, *Sessions*, *ManagingRoles*, *UserPrivileges*, *AccountManagement*, *TransferFund*, and *BalanceEnquiry*. Figure 3.7 is extended for the visualization of security patterns. Extended UML class diagram along with the visualization of security patterns is represented in Figure 3.8. Explanation of security patterns as visualized in extended UML class diagrams and how these security pattern help in achieving the security goals is explained in the following paragraphs.

For an online banking system, customer is the external entity to interact with the system. To provide clearly defined entrance to all the external entities

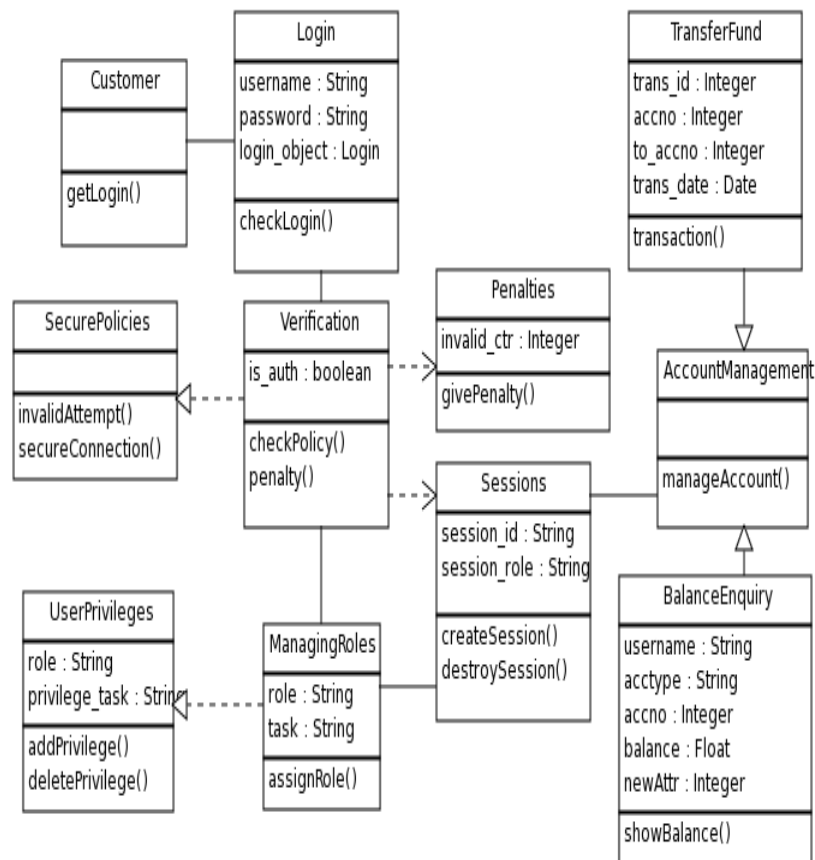


FIGURE 3.7: UML Class Diagram for Case Study

SAP(Single Access Point) security pattern is considered. *Customer* class plays the role of *ExternalEntity* which is a participant of *SingleAccessPoint* security pattern. Therefore, stereotype notation for *Customer* class is *Customer* {SAP / *ExternalEntity*} which is represented as 'CLASSNAME {PATTERN NAME/ROLE NAME}'. Customer opens the login screen to enter the system which is the only entry point to the system. Accordingly, stereotype notation for *Login* class is {SAP / *Singleton*}.

Customer authenticates itself by providing his required authentication information, this information is used for the verification of customer identity. *Verification* class verifies this information and authenticates the user depending on the security policies enforced by the system. *CheckPoint* security pattern is used for implementing security policies as required by the system and it is also used for penalizing the user for violating security policies. Verification class also plays the role of *InternalEntity* under the security pattern *SingleAccessPoint*. After the

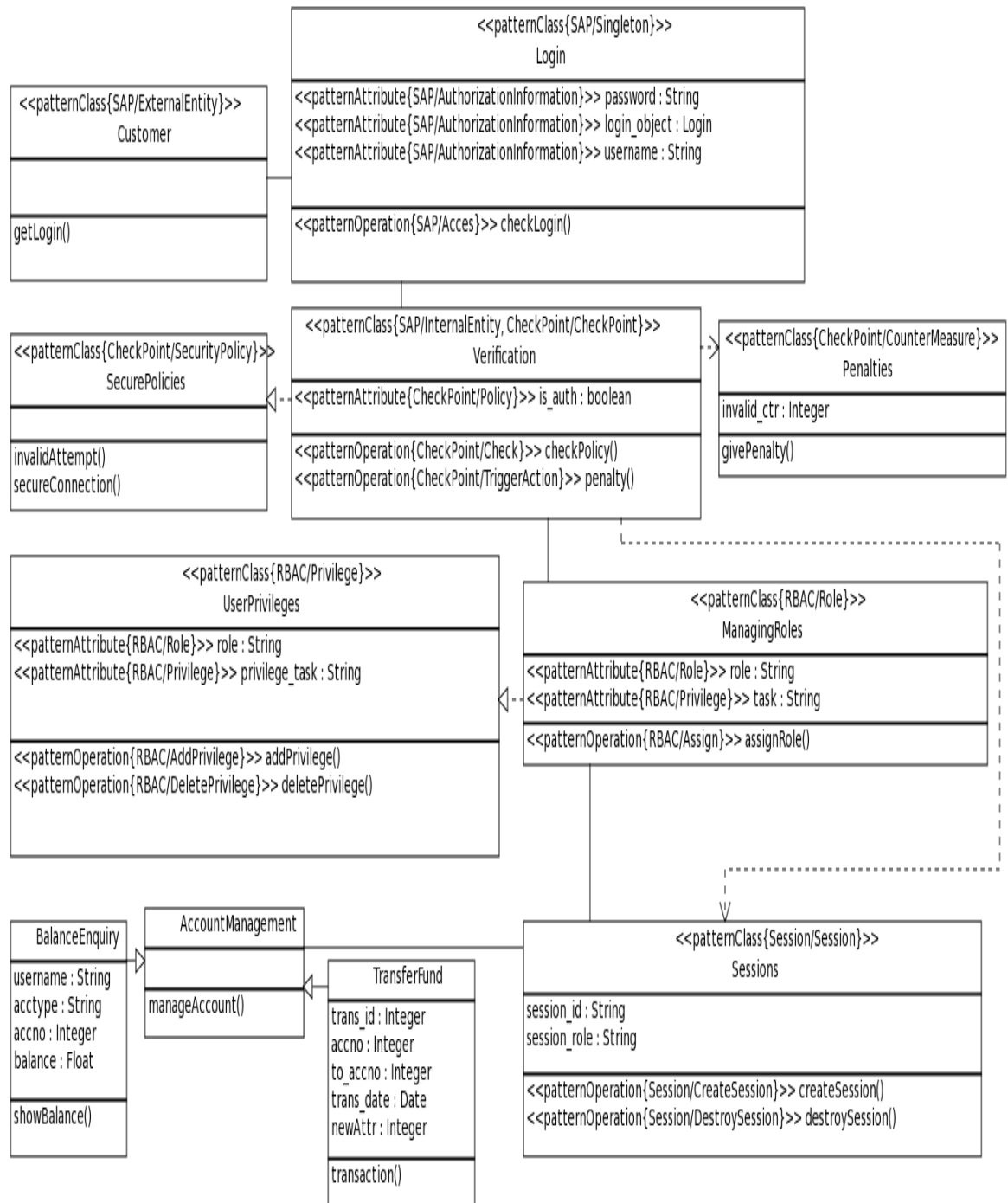


FIGURE 3.8: Extended UML class Diagram for Case Study


addition of roles, stereotype annotation for *Verification* class becomes *Verification* {*SingleAccessPoint/ InternalEntity, CheckPoint/CheckPoint*}.

User authentication is checked in *Verification* class and if the user is not identified, then method of *Verification* class triggers an action to impose penalty. The *Penalties* class performs a role of *CounterMeasure* under the pattern *CheckPoint*. Stereotype annotation of *Penalties* class becomes {*CheckPoint/CounterMeasure*}. After the authentication of user, system needs to identify the authorized area and restricted area for identified user, for this purpose RABC (Role Based Access Control) security pattern is used. When user is authenticated, its role is retrieved from the class *ManagingRoles* which plays the role of *Role* under the *Role Based Access Control* security pattern and its authorized area is retrieved from class *UserPrivileges* which plays the role of *Privilege* user the security pattern *RBAC*. Class which plays the role of *Privilege* must be associated with the class which plays the role of *Role* under the security pattern *RBAC*.

After the verification and recognition of the role and privileges of user, session must be created to store the global variables in order to keep track of the user identification information such identity, role and privilege. All other classes developed for handling actions such as transfer, withdrawal, deposit are supposed to be attached to session class, because session contains the global variables which hold information about the role and privileges of user. *Session* security pattern has been used for creating session and for storing global variables in order to secure the restricted areas. *Sessions* class performs the role of *Session* under the *Session* security pattern. All the other classes such as *BalanceEnquiry, AccountManagement, TransferFund* play the role of system component which uses sessions.

The above details show, how the four selected security pattern are helpful in archiving desired security goals. Every system which aims at providing a single entry point, user authentication, role and privileges for user, and needs to maintain session, can be made secure at the time of system design by applying four selected security patterns according to the pattern language shown in Figure 1.5.

The association text file is generated generate from Extended UML class diagram with security patterns is the parsed in the grammar and the class Association file generated looks like as given in Figure 3.3. Generated class association file is verified according to the grammar developed for pattern language shown in Figure 1.5.



```

C:\Windows\system32\cmd.exe - grun  securepattern classDecl -tree -gui grammar.txt
C:\antlr\security\newtest>java org.antlr.v4.runtime.nisc.TestRig securepattern c
lassDecl -tree -gui grammar.txt
(classDecl (className customer) (sapExternal ( (patternSAP SAP) / (roleSAPExtern
al ExternalEntity) ) (classSAPSingleton I (className Login) (sapSingleton ( (pat
ternSAP SAP) / (roleSAPSingleton Singleton) ) (classCheckPoint I (className Veri
fication) (checkPoint ( (patternCheckPoint CheckPoint) / (roleCPCheckPoint Check
Point) , (patternSAP SAP) / (roleSAPInternal InternalEntity) ) (classSecurityPol
icy I (className SecurePolicies) (securityPolicy ( (patternCheckPoint CheckPoint
) / (roleSP SecurityPolicy) ) (classCounterMeasure (className Penalties) (counte
rMeasure ( (patternCheckPoint CheckPoint) / (roleCM CounterMeasure) ) (classSess
ion (className Sessions) (session ( (patternSession Session) / (roleSession Sess
ion) ) (classRole (className ManagingRoles) (rbac ( (patternRBAC RBAC) / (roleR
OLE Role) ) (classPrivilege I (className UserPrivileges) (rolepriv ( (patternRBAC
RBAC) / (rolePrg Privilege) ) (finalsessionclass (className Sessions) (finalsess
ion ( (patternSession Session) / (roleSession Session) ) I (endclass I I )>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

```

FIGURE 3.9: Result - ANTLR Command Prompt

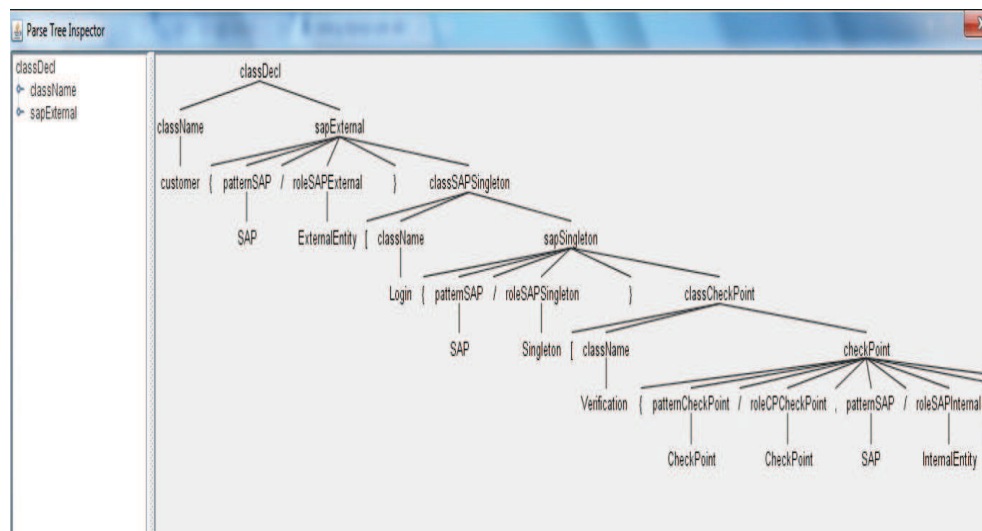


FIGURE 3.10: Result - ANTLR Parser Tree

3.2.1 Test Cases

In order to explain the verification process, we have considered three test cases as shown in the Figure 3.11, Figure 3.12, and Figure 3.13. These test cases are the class associativity text files generated from the class text of extended UML class diagram.

First test case shown in Figure 3.11 is generated from the extended UML class diagram which is shown in Figure 3.8. This test case is accepted by the proposed grammar because it strictly follows the pattern language shown in Figure 1.5.

```
Customer{SAP/ExternalEntity}
  [Login{SAP/Singleton}
    [Verification{CheckPoint/CheckPoint, SAP/InternalEntity}
      [SecurePolicies{CheckPoint/SecurityPolicy}
        Penalties{CheckPoint/CounterMeasure}
        Sessions{Session/Session}
        ManagingRoles{RABC/Role}
        [UserPrivileges{RABC/Privilege}
          Sessions{Session/Session}
          [AccountManagement{NULL/NULL}
            [BalanceEnquiry{NULL/NULL}
              TransferFund{NULL/NULL}
            ]
          ]
        ]
      ]
    ]
  ]
]
]
]
]
]
```

FIGURE 3.11: Test-Case 1

```
Customer { SAP/ExternalEntity, AbstractFactory/AbstractProduct}
  [Login { SAP/Singleton, AbstractFactory/createProduct}
    [ Verification{CheckPoint/CheckPoint, SAP/InternalEntity, Adapter/Adapter}
      [ SecurePolicies{CheckPoint/SecurityPolicy}
        Penalties{CheckPoint/CounterMeasure}
        Sessions{Session/Session, Factory/Creator}
        ManagingRoles{RABC/Role}
        [ UserPrivileges{RABC/Privilege}
          Sessions{Session/Session}
          [
            AccountManagement{AbstractFactory/AbstractFactory}
            [
              BalanceEnquiry{AbstractFactory/createProduct}
              TransferFund{AbstractFactory/createProduct}
            ]
          ]
        ]
      ]
    ]
  ]
]
]
]
]
]
```

FIGURE 3.12: Test-Case 2

```

Customer { AbstractFactory/AbstractProduct}
  [Login { SAP/Singleton, AbstractFactory/createProduct}
    [ Verification{CheckPoint/CheckPoint, Adapter/Adapter}
      [ SecurePolicies{CheckPoint/SecurityPolicy}
        Penalties{CheckPoint/CounterMeasure}
        Sesssions{Session/Session, Factory/Creator}
        ManagingRoles{RABC/Role}
      [ UserPrivileges{RABC/Privilege}
        Sessions{Session/Session}
      ]
    ]
  ]
]
]
]
]

```

FIGURE 3.13: Test-Case 3

Second test case shown in Figure 3.12 satisfies all the production rules according to the proposed grammar which is developed for pattern language shown in Figure 1.5. Therefore, it is accepted by the proposed grammar. Difference between the first and second test case is as follows: First test case does not contain any security pattern other than the four selected security patterns, for which the pattern language is composed. Second test case contains four selected security patterns as well as other Gamma et.al. design patterns but at the same time it is in accordance with proposed pattern language as shown in Figure 1.5.

Third test case shown in Figure 3.13 will not be accepted by the proposed grammar. Because, the very first line containing the class declaration of *Customer* class does not contain the pattern $\{SAP/ExternalEntity\}$, also the *Verification* class must contain the pattern $\{SAP/InternalEntity\}$ along with the pattern $\{CheckPoint/CheckPoint\}$ in order to comply with the proposed language. The result generated by using improved grammar is shown in Figure 3.9 and, Figure 3.10. Figure 3.9 shows the command prompt output. Figure 3.6 shows the parser tree generated by ANTLR tool.

Chapter 4

Selection of Appropriate Security Patterns

4.1 Modeling of Security Pattern for Building Repository

The first step in this approach is the creation of pattern repository. Creation of pattern repository is done by formalizing security patterns using Goal Oriented Requirement Language (GRL). The analyzation and decomposition of attributes, relationships and various influences of all design/security patterns is done in few steps and included in the repository in an orderly fashion. GRL model shows the contribution that a security pattern make on the security related Non Functional Requirements (NFRs), it identifies which NFR will be built and which NFR will be hurt by the use of particular security pattern. GRL modelling also helps in visualizing the different relationship among the patterns such as, whether a pattern can co-exist with other patterns and what are the prerequisite patterns for the particular pattern. Figure 4.1 shows intentional elements of GRL used for modeling different attributes of security pattern. In GRL tasks are represented by hexagonal shape and Soft Goals are represented by a cloud like curvilinear shape. In this study tasks are modeled as security patterns and Soft Goals are modeled as Non Functional Requirements (NFRs), contribution links are used for specifying the contribution of security pattern towards a soft goal along with the strength, and decomposition links are used for representing the relation among different patterns. Strengths are specified numerically, Make (1.00), Help (0.75), Unknown (0.50), Hurt (0.25), Break (0.00). The four well know architectural security patterns(Single Access Point, Security Session, Role-Based Access Control, Check Point) proposed by Yoder has been considered to demonstrate proposed approach.





GRL Intentional Elements	Intention	Security Pattern Modelling
	Tasks	Specifies Security Pattern
	Soft Goal	Represents Non Functional Requirements(NFRs)
	Contribution Link	Shows Make and Hurt contribution of Security pattern towards the NFRs
	Decomposition Link	Shows the relation between different patterns

FIGURE 4.1: GRL intentional elements

Implications of Patterns proposed by Yoder:

- Single Access Point helps in building Integrity, Confidentiality, and Accountability, at the same time Single Access Point hurts the Availability of system. Single Access Point also depends on Check Point for its existence.
- Role-Based Access Control helps in building Manageability, Availability, Integrity and Confidentiality.
- Security Session helps in building Availability, Integrity, Confidentiality, Accountability and Usability.
- Check Point security helps in building Confidentiality, Integrity, Availability, same forces are also built by Security Session and RBAC.

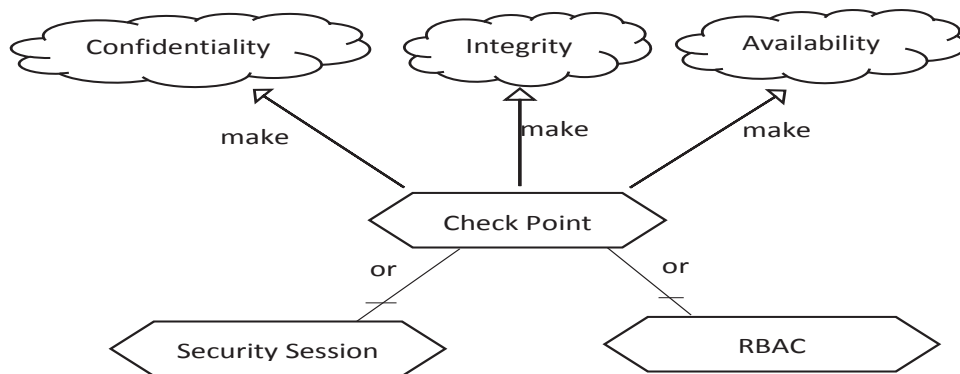


FIGURE 4.2: GRL model of Check Point Security Pattern

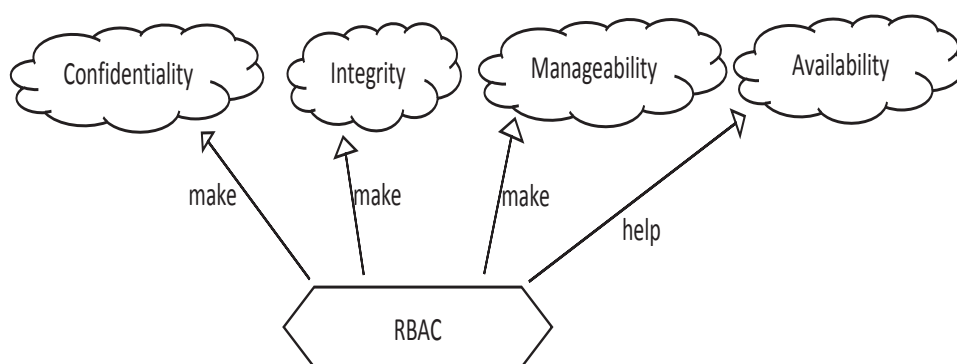


FIGURE 4.3: GRL model of Role-Based Access Control Security Pattern

GRL model for Check Point and Role Based Access Control scrutiny pattern is shown in Figure 4.2 and 4.3 respectively [32]. In these figures Security Patterns Check Point and Role Based Access control are represented using hexagonal

shape know as Task in GRL, tasks are connected with Non Functional Requirements(NFRs) through the contribution links. Contribution links can be marked as make, help, unknown, hurt and break, this helps in deciding the strength by which security pattern affect the connected Non Functional Requirement. Relation among the security patterns is represented with the help of decomposition link, it shows the relation between the two tasks. Decomposition links can be marked as 'and' and 'or', 'and' contribution is positive and necessary and 'or' contribution is positive but not necessary.

4.1.1 Extraction of Facts from GRL Model

Fact are extracted from Goal Oriented Requirement Language (GRL) with the help of XML and represented in the form of instance of relational algebra. Five instances were built in order to store the security patterns.

- Instance P for storing the security pattern, it consists of two attributes 'patternid' and 'patternname'.
- Instance NFR for storing the Non Functional Requirements (NFR), it consists of two attributes 'nfrid' and 'nfrname'.
- Instance F for storing the Non-Functional Requirement and 'patternid' for each NFR's affected by a particular security pattern along with the 'strength' by which pattern contributes to the corresponding NFR.
- Instance R for storing the relation among the pattern in order to return the list of prerequisite patterns.

TABLE 4.1: Instance P

patternid	patternname
1	Single Access Point
2	Role-Based Access Control
3	Security Session
4	Check Point

TABLE 4.2: Instance NFR

nfrid	nfrname
1	Confidentiality
2	Integrity
3	Availability
4	Accountability
5	Usability
6	Manageability

TABLE 4.3: Instance F

patternid	nfrid	strength
1	2	.75
1	1	.75
1	4	.75
1	3	.25
2	6	1
2	3	.75
2	2	1
2	1	1
3	3	.75
3	2	.75
3	1	.75
3	4	.75
3	5	.75
4	3	1
4	2	1
4	1	1

TABLE 4.4: Instance F

patternid1	patternid2	relation
4	3	or
4	2	or
1	4	and

4.1.2 Selection of Appropriate Security Pattern that Fulfills Security Requirement

Selection of the appropriate security pattern is done with the help of queries made to the instances. For this purpose another instance Goal is created containing the ID(bfrid) of the Non Functional Requirements (NFR) along with the required

strength.

$$Fullfilled = \sigma_{strength>Goal.s} \quad \text{and} \quad nfrid=Goal.nfrid(F) \quad (4.1)$$

$$Tempfulfilled = \Pi_{nfrid}(Fullfilled) \quad (4.2)$$

$$NotFullfilled = (\Pi_{nfrid}(G) - Tempfulfilled) \quad (4.3)$$

$$PatternFullfiled = (Fullfilled \bowtie P) \quad (4.4)$$

$$TmpPrerequisite = (\Pi_{patternid}(PatternFullfiled)) \quad (4.5)$$

$$Fullfilled = \sigma_{patternId=TmpPrerequisite.patternid}(R) \quad (4.6)$$

$$TmpPrerequisite = (\Pi_{patternname}(\Pi_{patternid2}(Tmp2Prerequisite))) \bowtie P \quad (4.7)$$

Query 'Fulfilled' extract the 'patternid' of all the pattern which satisfies the NFR's with the strength specified in the instance 'Goal'. 'Tempfulfilled' extract the 'nfrid' of the fulfilled NFR leaving behind the unfulfilled NFR's. Subsequently, 'NonFulfilled' extract the unfulfilled NFR's by subtracting the 'Tempfulfilled' from instance the 'Goal'. Now finally to extract the name of satisfying pattern a join is made between instance 'P' and 'Fulfilled' which is then stored in 'PatternFullfiled'. Last step is to extract the dependencies in order to find the prerequisite patterns for the selected patterns. These dependencies are extracted with the help of query 'Prerequisite', it will extract the name of perquisite security patterns. Proposed relational algebra can be implemented on any relational database. In this study we made an attempt to develop a service which will allow user to select appropriate security pattern that fulfills the required nonfunctional requirements.

Chapter 5

Detection of Security Patterns

5.1 Detection of Security Patterns

Software maintenance is known as the most expensive one among all the phases of software development life cycle, both in terms of money and time. Reverse engineering activities are adopted to maintain, evolve and re-engineer the system. Design pattern detection and software architecture reconstruction are the most important fields of reverse engineering. Design pattern detection aims at identification of design patterns that have been used in the implementation of a particular system, whereas, software architecture reconstruction lets the software engineers view different levels of abstractions of the system. Hence, the reverse engineering techniques let user focus on the overall architecture of system without having adequate knowledge of detailed programming implementation. Security patterns detection helps developers to find the security constrains that were focused, during the development of system. Thus strengthening and helping in maintenance of security related Non Functional Requirements of system.

An Eclipse plug-in i.e., MARPLE (Metrics and Architecture Reconstruction Plug-in for Eclipse) has been developed by Fontana Et al.[40], which supports both design pattern detection and Software architecture Reconstruction. They have tried to improve the accuracy of result with the help of various classifiers.

5.2 Implementation

5.2.1 Hybrid Classifier

Many research have been done and it has been proven that use of hybrid classifiers produce more accuracy than the usual ones [41] [42] [43]. Naive Bayes induction algorithms were proven to be significantly correct on many classification tasks irrespective of assumptions made on conditional independence. Again, the correctness of Naive-Bayes is not as promising as Decision Tree classifiers. Hence, a new hybrid classifier DTNB(Decision Tree Naive-Bayes), which induces a hybrid of Naive-Bayes classifiers and decision-tree classifiers is chosen. The leaves contain Naive-Bayesian classifiers but the decision-tree nodes contain univariate splits as regular decision-trees. The approach helps in retaining the interpretability of Decision Tree and Naive-Bayes. Therefore, resulting in improved classifiers

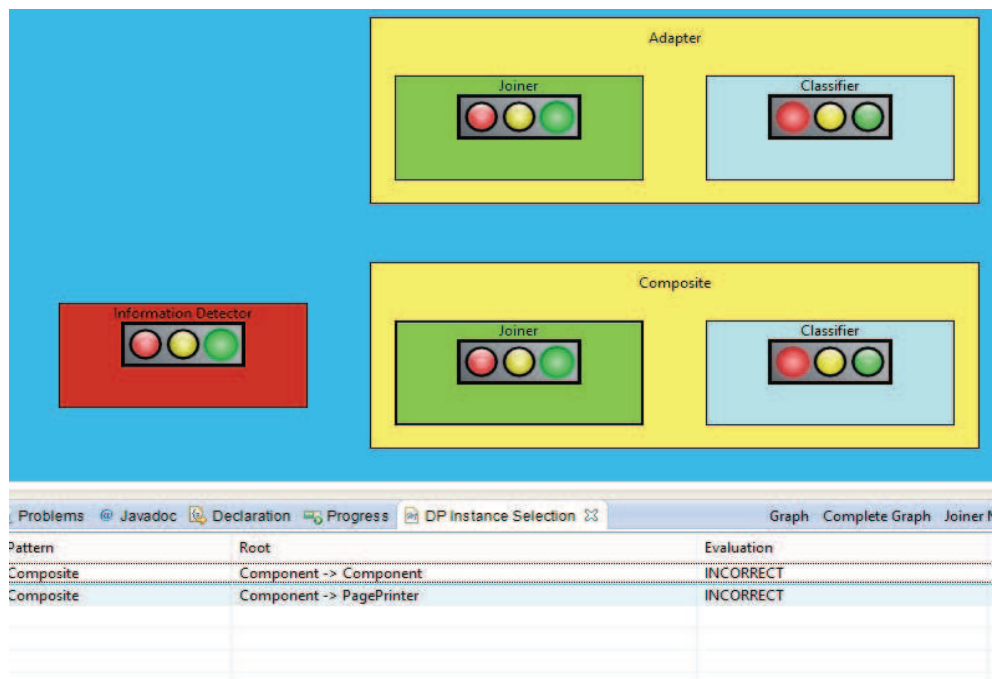


FIGURE 5.1: jRefractor - Composite Pattern Detection Result

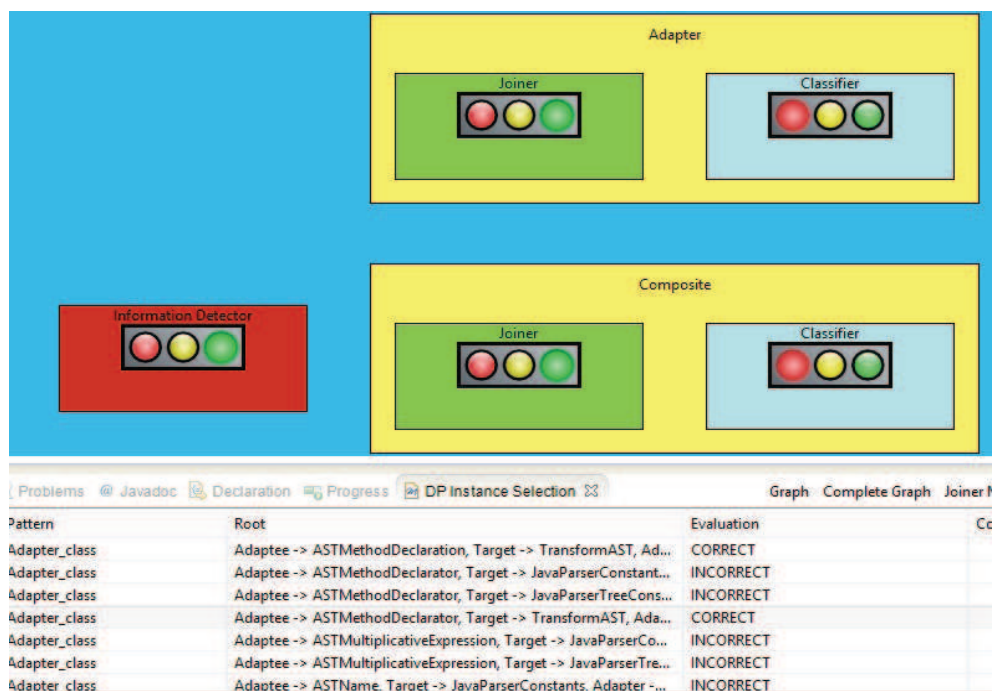


FIGURE 5.2: jRefractor - Adapter Pattern Detection Result

which frequently outperform both constituents, this classifiers shows drastically improved performance especially in the larger databases. Result obtained using MARPLE are shown in Table 5.1

TABLE 5.1: Result Using Similarity Score

Classifier	Accuracy for Singleton
NaiveBayes	0.81
ZeroR	0.61
OneR	0.87
RandomForrest	0.93
DTNB	0.89

Application of DTNB instead of Naive-Bayes led to only a slight increase in the accuracy of detection. Therefore similarity score method is used for the detection of Security Patterns.

5.2.2 Detection Using Similarity Score

Similarity algorithm depends on the system graph size for convergence. Time required for calculation of similarity score between pattern graph and all the vertices of system graph gradually increases as the number of vertices of the graphs increase [39]. In order to make the matching more efficient, graph size must be reduced without losing vital information for design pattern detection process. Since most of design patterns involve hierarchies, similarity algorithm can be applied to classes which involve inheritance structures. If a role of class is assigned a score which is less than the score of role of another class, then the class will the lower score should satisfy the fewer criteria as described in the pattern. The Similarity Score Matrix S was calculated using the following algorithm [39].

1. Set $Z_0 = 1$
2. Iterate the below equation even number of times

$$Z_{k+1} = \frac{BZ_k A^T + B^T Z_k A}{\|BZ_k A^T + B^T Z_k A\|}$$
3. Last value of Z_k will give the Similarity Score Matrix Where

- A is the adjacency matrix for graphs G_A and B is the adjacency matrix of graph G_B

- Initially Z_0 is filled with ones.

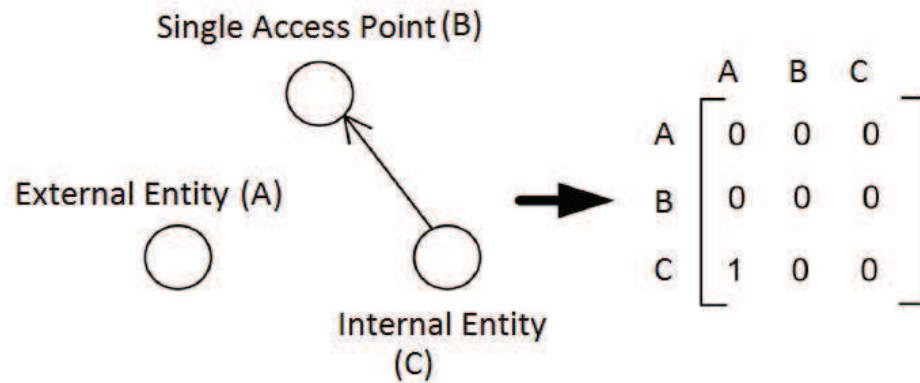


FIGURE 5.3: Assosiation Matrix For Single Access Matrix

Association graph and Association matrix for Single Access Point Security Pattern is shown in Figure 5.3. Similarity score method was applied on open source software JhotDraw and JRefactory. Result obtained is show in Table 5.2.

TABLE 5.2: Result Using Similarity Score

Security Pattern	JhotDraw	JRefactory
Single Access Point	2(TP) 0(FN)100(Recall)	12(TP) 0(FN) 100(Recall)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

6.1.1 Formalization and Validation of Security Patterns

Security is a critical issue which can't be only based on UML diagrams, hence it requires proper verification in terms of formal language. In this study, an attempt has been made to propose a grammar which satisfies the security pattern language and formally verifies the security patterns. In order to demonstrate this approach, a case study on online banking system has been considered. For this case study, extended UML class diagram visualizing security patterns is generated by using UML class diagram. Single Access Point, provides a single login screen to all external entities of the system, which helps the system to trace the unusual requests thus maintaining the availability of the system for other entities. Check Point ensures the confidentiality of system by authenticating the user and it also enforce certain security policies and penalizes the user for violating security policies. The role-based access control (RBAC) maintains the integrity of the system authorizing the user with the help of user role-privilege relationship. RBAC also improves the confidentiality of the system by providing access rights.

6.1.2 Selection of Appropriate Security Patterns

Formalization of security patterns has been done in order to create a repository. Queries are made to repository in order to find the most appropriate security pattern for the set of given security related Non Functional Requirements. This approach not only find list of most appropriate security patterns but it also check for the dependencies among the patterns in order to find the prerequisite patterns. With the help of GRL security pattern were formalized subsequently facts were extracted from the formalized security patterns. Modelling security patterns with the help of GRL allows to accurately and effectively describe how each patterns make a distinct contribution to a security related Non Functional Requirements. Facts extracted from GRL were represented in form of instances for relational database. For finding list of appropriate security pattern and prerequisite pattern, queries written using relational algebra were made to the instances. Thus making the following contributions: (i) relational algebra have well found semantics; hence used for modelling the data stored in relational databases. Therefore this

approach can be implemented as service using any relational database server. (ii) relational databases, such as MySQL can be easily optimized even if the number of security patterns gradually increases, where else on the other hand client side language performance will decrease if the number of security patterns will gradually increase. (iii) in client side languages when the number of security patterns will increase it will also lead to the increase in size of repository which will make it difficult to distribute, where else on the other hand in this approach, repository is stored in server and an interface for making query to the server is provided. (iv) this approach will help in creating a centralized pool of security patterns, where all the available security patterns are stored in the repository on the server. Security Patterns Search Engine[44] was developed by using this approach. As a result, security patterns were formalized which help in identifying the implications and liability imposed by patterns which are not easy to identify in case of textual representation, approach for finding appropriate security patterns and corresponding prerequisite patterns with the help of relation algebra has been proposed.

6.1.3 Detection of Security Patterns

This method provides a approach to o detect security patterns in the source code of a software. In the field of Software Reverse Engineering, this approach to detect security pattern instances in a software, is quite adoptable as it automatically detects design patterns. The use of Similarity Score method in the process of design pattern detection provides a way not only to detect full occurrence of the pattern but it also provides a measure to find the percentage matching of the pattern. This method is useful for software engineers to get knowledge about the pattern existence in the system. MARPLE has been to extract the security pattern candidate from the source code of the system.

6.2 Future Work

A good number of security patterns can be added in order to extend the pattern language. As per now, the association of classes is being checked. Grammar can be extended for verifying the operations of class and role played by these operations under security pattern. It means if a class contains some operation playing some role under security pattern , then grammar can be developed to verify whether this

class declaration satisfies the pattern language or not. This will help developer to resolve security issues at development stage itself, thus leading to rapid development of software and saving lots of time invested in testing. Functionalities of proposed approach can be developed as an extension for widely used UML drawing software solutions such as IBM Rational Rose. Basic fundamental advantage of security patterns is reusability, for this purpose XML file can be generated and saved for further use, also skeleton source code can be generated out of the UML class diagram for several programming languages.

Bibliography

- [1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [2] Deepak Alur, Dan Malks, John Crupi, Grady Booch, and Martin Fowler. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Prentice Hall, 2nd edition, 2003.
- [3] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, Boston, USA, 2002.
- [4] Frank Buschmann, Kelvin Henney, and Douglas Schimdt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Language*, volume 4. John Wiley & Sons Ltd., West Sussex, England, 2007.
- [5] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, West Sussex, England, 2005.
- [6] Christopher Steel, Ramesh Nagappan, and Ray Lai. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall PTR, 2005.
- [7] Ashish Kumar Dwivedi and Santanu Kumar Rath. Incorporating security features in service-oriented architecture using security patterns. *ACM SIGSOFT Software Engineering Notes*, 40(1):1–6, 2015.
- [8] Jörg Niere, Wilhelm Schäfer, Jörg P Wadsack, Lothar Wendehals, and Jim Welsh. Towards pattern-based design recovery. In *Proceedings of the 24th international conference on Software engineering*, pages 338–348. ACM, 2002.

-
- [9] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. In *In proceeding of the 4th Conference on Patterns Language of Programming (PLoP'97)*, 1997.
- [10] Robert Hanmer. *Patterns for fault tolerant software*. John Wiley & Sons, 2007.
- [11] Jing Dong and Sheng Yang. Extending uml to visualize design patterns in class diagrams. In *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 124–131. San Francisco Bay, California, USA, 2003.
- [12] Jing Dong, Sheng Yang, and Kang Zhang. Visualizing design patterns in their applications and compositions. *IEEE Trans. Softw. Eng.*, 33(7):433–453, July 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.1012. URL <http://dx.doi.org/10.1109/TSE.2007.1012>.
- [13] Hong Zhu and Ian Bayley. An algebra of design patterns. *ACM Trans. Softw. Eng. Methodol.*, 22(3):23:1–23:35, July 2013. ISSN 1049-331X. doi: 10.1145/2491509.2491517. URL <http://doi.acm.org/10.1145/2491509.2491517>.
- [14] Toufik Taibi and David Chek Ling Ngo. Formal specification of design patterns - a balanced approach. *Journal of Object Technology*, 2(4):127–140, 2003.
- [15] Linda Rising. *The pattern almanac*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [16] Scott Henninger and Victor Corrêa. Software pattern communities: Current practices and challenges. In *Proceedings of the 14th Conference on Pattern Languages of Programs*, page 14. ACM, 2007.
- [17] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.
- [18] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 21–26. IEEE, 2007.
- [19] John Vlissides. Notation, Notation, Notation. C++ Report. Technical report, April 1998.

-
- [20] Jing Dong. Uml extensions for design pattern compositions. *Journal of object technology*, 1(5):151–163, 2002.
- [21] Stefan Berner, Martin Glinz, and Stefan Joos. A classification of stereotypes for object-oriented modeling languages. In *Proceedings of the Second International Conference on the Unified Modeling Language (UML), LNCS1723*,, pages 249–264. Springer-Verlag, October 1999.
- [22] Jing Dong, Tu Peng, and Yajing Zhao. Automated verification of security pattern compositions. *Information and Software Technology*, 52(3):274–295, 2010.
- [23] Ashish Kumar Dwivedi and Santanu Kumar Rath. Analysis of a complex architectural style c2 using modeling language alloy. *Information and Software Technology*, 3(2):152–164, 2014.
- [24] Ian Bayley and Hong Zhu. Formal specification of the variants and behavioral features of design patterns. *Journal of Systems and Software*, 83(2):209–221, 2010.
- [25] Ashish Kumar Dwivedi and Santanu Kumar Rath. Selecting and formalizing an architectural style: A comparative study. In *Contemporary Computing (IC3), 2014 Seventh International Conference on*, pages 364–369. IEEE, 2014.
- [26] Shouvik Dey and Swapan Bhattacharya. Formal specification of structural and behavioral aspects of design patterns. *Journal of Object Technology*, 9(6):99–126, 2010.
- [27] Arturo Hinojosa and Joshua Brett Tenenbaum. A cognitive model of design pattern selection. *Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology*, 2004.
- [28] Hervé Albin-Amiot, Pierre Cointe, Y-G Guéhéneuc, and Narendra Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 166–173. IEEE, 2001.
- [29] Siani Pearson and Yun Shen. Context-aware privacy design pattern selection. In *Trust, Privacy and Security in Digital Business*, pages 69–80. Springer, 2010.

-
- [30] Jing Wang, Yeong-Tae Song, and Lawrence Chung. From software architecture to design patterns: A case study of an nfr approach. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on*, pages 170–177. IEEE, 2005.
- [31] Gunter Mussbacher, Michael Weiss, and Daniel Amyot. Formalizing architectural patterns with the goal-oriented requirement language. In *Nordic Pattern Languages of Programs Conference*, 2006.
- [32] Michael Weiss and Haralambos Mouratidis. Selecting security patterns that fulfill security requirements. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 169–172. IEEE, 2008.
- [33] Lutz Prechelt and Christian Krämer. Functionality versus practicality: Employing existing tools for recovering structural design patterns. *Journal of Universal Computer Science*, 4(11):866–882, 1998.
- [34] Lothar Wendehals. Improving design pattern instance recognition by dynamic analysis. In *Proc. of the ICSE 2003 Workshop on Dynamic Analysis (WODA), Portland, USA*, pages 29–32, 2003.
- [35] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.
- [36] Giuliano Antoniol, Gerardo Casazza, Massimiliano Di Penta, and Roberto Fiutem. Object-oriented design patterns recovery. *Journal of Systems and Software*, 59(2):181–196, 2001.
- [37] Zsolt Balanyi and Rudolf Ferenc. Mining design patterns from c++ source code. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 305–314. IEEE, 2003.
- [38] Columbus Reverse Engineering Tool. <https://frontendart.com/>, 2006.
- [39] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

- [40] Francesca Arcelli Fontana and Marco Zanoni. A tool for design pattern detection and software architecture reconstruction. *Information sciences*, 181(7):1306–1324, 2011.
- [41] Peter D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of artificial intelligence research*, pages 369–409, 1995.
- [42] Rajat Raina, Yirong Shen, Andrew McCallum, and Andrew Y Ng. Classification with hybrid generative/discriminative models. In *Advances in neural information processing systems*, page None, 2003.
- [43] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.
- [44] Mohd Suleman. Security patterns search engine. <http://computerinfo.in/securityps/>, 2014.