2003

Architecture-level risk assessment tool based on UML specification

Tianjian Wang
West Virginia University

Architecture-level Risk Assessment Tool Based on UML Specification

Tianjian Wang

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Hany Ammar , Ph.D., Chair
K. Goseva-Popstojanova, Ph.D.
Gamal Fahmy, Ph.D.

Lane Department of Computer Science & Electrical Engineering

Morgantown, West Virginia University
2003

Keywords: Risk Assessment, Dynamic Matrix, Software Engineering

ABSTRACT

Architecture-level Risk Assessment Tool Based on UML Specification

Tianjian Wang

Most faults in software systems are likely to be found in only a few of components [1]. The early identification of these components allows the project management to focus on remedial actions, such as redesigning the critical components that are likely to cause field failures or optimally allocating resources on implementation and testing [2]. This thesis presents a prototype tool called **Architecture-level Risk Assessment Tool** (ARAT) to demonstrate the process of risk assessment. The final result of this process is to distinguish those potentially high risk components in the software system. ARAT is built on the risk assessment methodology [3]. By manipulating the data acquired from domain expert and measures obtained from Unified Modeling Language (UML) artifacts [4], ARAT can be used in the design phase of the software development process to improve the quality of the software product. A paper which demonstrates this tool is also published [19].

Dedication

I am honored to dedicate this paper to all the members of my family, who have encouraged me, and supported me throughout my life. I want to specifically express my love and appreciation to my lovely and beautiful sister, the one who shares my burden and dream, stress and joy.

Acknowledgements

First, I would like to express my deepest gratitude and appreciation to my research advisor, Dr. Hany Ammar, for this opportunity he gave me to conduct research under his supervision, for his ever presence guidance during this research effort and the freedom he give me to learn and explore.

I would like to thank Dr. Katerina Goseva-Popstojanova for her support and review and for serving as a member of my graduate committee.

I would like to thank my research colleagues, especially Ahmad Hassan, for the expertise he provided through out this research effort.

I would also like to thank Dr. Gamal Fahmy for taking time to be a member of my graduate committee and review this document.

This work is funded in part by grants to West Virginia University Research Corp. from the National Science Foundation Information Technology Research (ITR) Program grant number CCR-0082574 and from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia.

TABLE OF CONTENTS

Lists of Figures

CHAPTER 1: INTRODUCTION

1.1 What is ARAT

This tool (ARAT) is an implementation of the methodology presented in publication [3]. It uses quantitative metrics to systematically evaluate the quality of the software architecture. It also integrates the real time failure probability estimation, and the severity metrics calculation into the risk assessment model.

1.2 Problems and Solutions

Problems exist in current software engineering quality assurance applications. Many quality assurance methods or tools are applied in the late phase of the software life cycle. Due to some important product quality characteristics, like performance, reliability, maintainability, which can not be added in the late phase of the software lifecycle, any corrections made in the earlier phases on defect would be cost-effective. Otherwise, the failure would be expected when the requirement must be satisfied. Hence, early warnings and corrective activities of poor quality software product would be strongly desired for effective quality assurance. In addition, software architecture describes both the static structure and the dynamic behavior of the software. It is the key in software design and software quality analysis. As a solution to the problems, our **Architecture-level Risk Assessment Tool (ARAT)** is created to track the quality of software product. Because the risk analysis is based on measurements and calculations of the high-level design diagrams, ARAT can be used as early as in the design phase of the software development process. It measures dynamic metrics proposed in [2] and further analyzes the quality of the architecture to produce architectural-level software risk assessment [3].

1.3 Related work and Objectives

Some current tools in the market doing the risk assessment are based on the source code of the software, [8] they first obtain the static metrics from source code, and then go on carrying out risk analysis on these metrics. But source code metrics are affected by the programming style of the programmer, as well as the programming language itself with its structures affecting the metrics results. When calculating the metrics from architectural descriptions like UML, we achieve independence of languages and human factors [9].

What is more, it is strongly desired by the project management to acquire the result of the risk assessment for the target system as early as possible. It would be impossible or resource wasteful to correct the error if we have to wait to get the result after part or full implementation is finished during the lifecycle of the software development. ARAT that examines UML at design phase has obvious advantages over those tools built on source code.

On the other hand, some tools [10] do get description from intermediate file by using certain CASE tools; they can be used in design phase as well, but they only produce static metrics to describe the model with limited capability, which is not enough to accurately represent the dynamic behavior of the architecture. They even require the output of result in a specific chosen format which is not convenient for popular use, some tools require extra information saved in a file which is not directly acquired from the model to describe the target software system model, then measurement and analysis based on the information would not be precise. As a result, it is not suggested to be adopted widely. Under this circumstance, we simply access the result of a CASE tool to carry on the risk analysis. The result is in general textual format and obtained directly from UML model diagrams of the target software system by running a very simple script. All further steps of analysis are based on this result. Thus, we not only achieve the accuracy and performance of the analysis, but also have the very straightforward

way to do the analysis. The simple, effective and practicable method with high performance is the contribution of this tool.

In summary, the main objectives of this tool are listed below:

1. It can carry on the risk assessment as early as the design phase in the lifecycle of the software development.
2. It can be compatible with popular design/modeling tools.
3. It is able to precisely compute the scenarios/use cases/system risk factors.
4. It is able to determine the distribution of the scenario/use case/system risk factors over different severity classes.
5. It is able to identify critical components based on the risk estimation.
6. It has user interface with less training and learning.
7. It contains high flexibility and extensibility to wrap more functional modules which will be developed in the future like performance analysis module, reliability analysis module, hazard analysis module etc.
8. The tool is portable and scalable.
9. Popular adoption and usage is feasible.

1.4 Preview of the chapters

The rest of this document is organized as follows: Chapter 2: describes the background to produce ARAT including a brief introduction on metrics, risk assessment as well as the methodology used by ARAT. Chapter 3: gives an overview of the whole system. Chapter 4: describes the design issue of the main ARAT components. Chapter 5: describes the implementation of various components in ARAT. Chapter 6: discuss the module verification and integration testing. Chapter 7: brings up a brief summary about this tool and discusses some future work on ARAT.

CHAPTER 2: BACKGROUND

The fundamental knowledge of metrics is necessary to understand risk assessment process. This chapter provides a brief introduction about metrics, risk assessment, and how to perform risk assessment based on the proposed methodology [3]. The benefit of adopting this methodology is also included.

2.1 The basic of metrics

Software metrics is any type of measurement which is related to a software system, process or related documentation [12]. Specifically, dynamic metrics are collected by measurements made of a program in execution while static metrics are collected by measurements made of the system representations such as the design, program or documentation. Dynamic metrics are fairly closely related to software quality attributes like the efficiency and the reliability of a program whereas static metrics help to assess the complexity, maintainability and other attributes of a program.

2.1.1 Connector and Component

A component can be as simple as an object, a class, or a procedure, and as elaborate as a package of classes or procedures. Connectors can be as simple as procedure calls; they can also be as elaborate as client-server protocols, links between distributed databases, or middleware. [3]

2.1.2 Dynamic Specifications Metrics

In order to estimate the fault proneness of software components and connectors, a dynamic metric for components is computed based on the dynamic complexity of

state chart [13]. The normalized dynamic complexity DOC_i^x of a component is calculated based on the following formula:

$$DOC_i^x = \frac{doc_i^x}{\sum_{k \in S_x} doc_k^x}.$$

Similarly, a dynamic metric for dynamic coupling between components is also computed for connectors [13]. The normalized dynamic coupling EOC_{ij}^x of a connector is based on the following formula:

$$EOC_{ij}^x = \frac{\left|MT_{ij}^x\right| \quad i, j \in S_x, i \neq j}{\left|MT^x\right|}.$$

The detail explanation of both formulas can be found in the methodology publication [3].

2.2 The basics of risk assessment

Before introducing the process of methodology used by ARAT, a quick review of some key terms in the field is necessary for the completeness of this thesis.

2.2.1 Risk defined in methodology

According to the NASA-STD-8719.13A standard [6], risk is a function of the anticipated frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity. This standard defines several types of risk, for example availability risk, acceptance risk, performance risk, cost risk, schedule risk, etc. Reliability-based risk is the only concern in our methodology, which takes into account the probability that the software product will fail in the operational environment and the adversity of that failure [3].

In the methodology [3] used by ARAT, risk is defined as a combination of two factors [11]: probability of malfunctioning (failure) and the consequence of malfunctioning (severity).

2.2.2 Performing risk assessment

Risk assessment is a useful means for identifying potentially troublesome software components that require careful development and allocation of more testing effort [1]. Risk assessment can be performed at various phases throughout the development process, for example, it could be based on an architecture model, an abstract design or implementation details etc. We believe that risk assessment being carried on at the architectural level is more beneficial than doing assessment at later development phases. Several reasons are shortly explained in section 1.2 and section 1.3.

2.3 Methodology

The complete introduction, calculation, and conclusion of the methodology from which ARAT is derived can be found in [3]. I only give an overview of the process of the methodology in this thesis.

2.3.1 Overview of the methodology

The methodology [3] uses dynamic complexity and dynamic coupling metrics that we obtained from the UML specifications. Severity analysis is performed using the Failure Mode and Effect Analysis (FMEA) technique. We combine severity and complexity (coupling) metrics to obtain heuristic risk factors for the components (connectors). Then, a Markov model is developed to estimate the scenario's risk factors from the risk factors of components and connectors. Further, use case and overall system risk factors are estimated using the scenario's risk factors.

2.3.2 Risk analysis process

The use cases and scenarios of a UML model drive the risk analysis process, the process iterates on the use cases and the scenarios that realize each use cases and determines the component/connector risk factors for each scenario, as well as the scenarios and use cases risk factors. For each scenario, the component/connector risk factors are estimated as a product of the dynamic complexity/coupling of the component/connector behavioral specification measured from the UML sequence diagrams and the severity level assigned by the domain expert using hazard analysis and Failure Mode and Effect Analysis. Furthermore, a Markov model is automatically constructed for each scenario based on the sequence diagram, thus, a scenario risk factor is determined. The methods for estimation of the use cases and overall system risk factors are given as well. The final result of the above process would be a list of critical use cases, a list of critical scenarios in each use case or a list of critical components/connectors for each scenario and each use case.

An assumption is made for this process that the UML architectural model consists of a use case diagram defining several independent use cases, and that each use case consists of one or more independent scenarios modeled by using sequence diagrams. The detail risk analysis process from the methodology is shown in Figure 1. The most inner rectangle contains actions conducted at the bottom level of the process, which calculates the dynamic complexity and dynamic coupling for components and connectors. After the domain expert specifies severity for each component and connector, the risk factor for every components and connectors are calculated. At this stage, feedback could be made immediately after identifying the critical components. However, it would be more practical to wait until the whole process is complete with the identification of critical component/connector in the system scope. Any modifications based on the feedback would change the risk

factor of each component/connector and further affect the rank of scenarios and use cases. The process is repeated until the specifications or requirements are satisfied and no further action is needed.

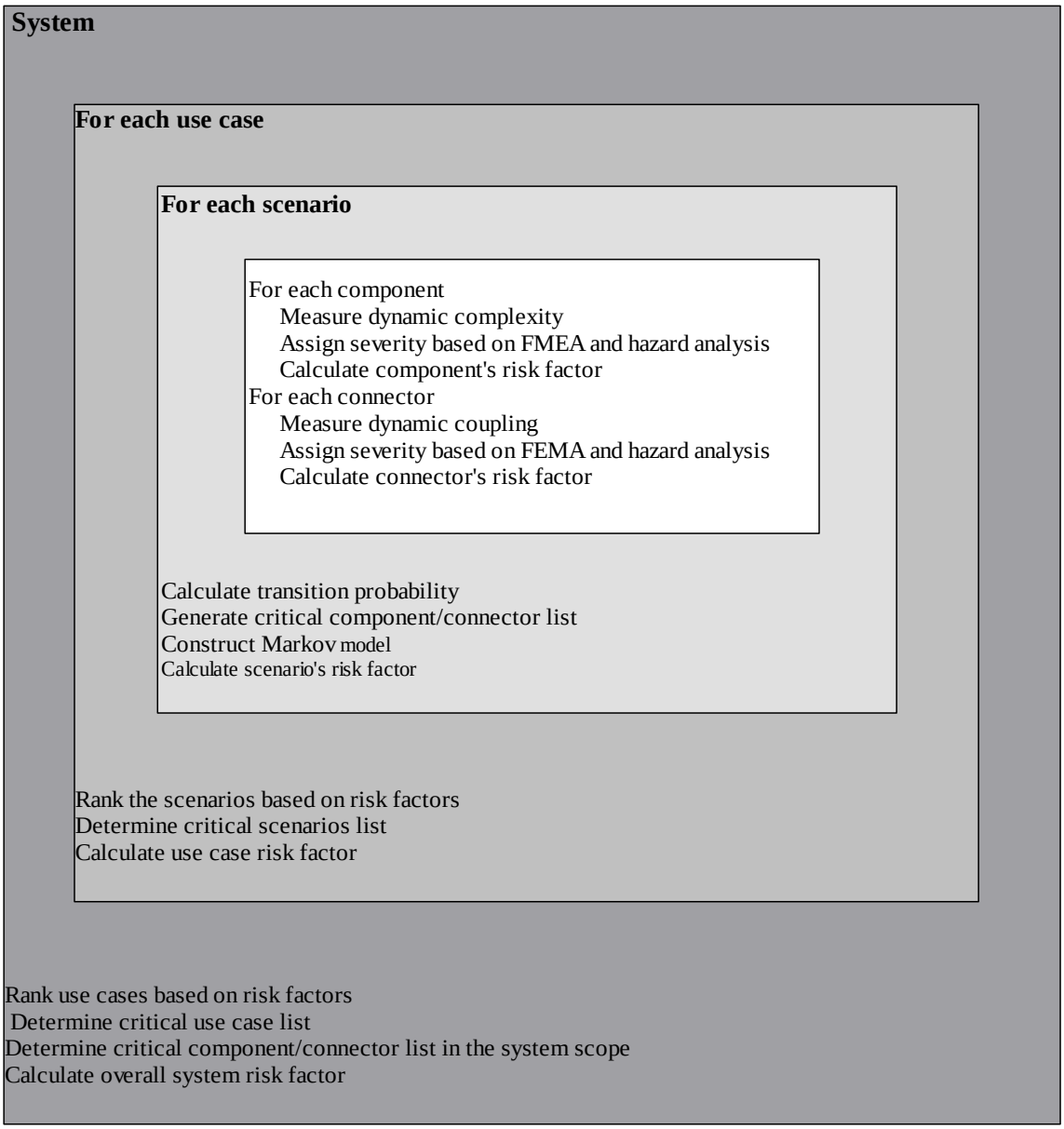


Figure 2: The Risk Analysis Process

CHAPTER 3. SYSTEM OVERVIEW

This chapter briefly explains how to build the ARAT system as well as how to reach the objectives and requirements of the system.

3.1 ARAT overview

Building the model of a software system will be conducted by a model tool and is not expected by ARAT. But it does intend to be compatible with the most popular model tool in the market if it's not feasible to be compatible with all of them. In order to satisfy the objectives of ARAT, the model information of the target software system must be converted to a suitable format for analysis at the beginning of the risk assessment process. Based on our observation and conclusion, Rational Rose RealTime [18] is the most popular model tool currently in the market, it would be a good candidate to work as the front end of ARAT. Meanwhile, we can take advantage of the extensibility interface embedded in it to transform the visual model diagram into textual format data directly. This can save a lot of hassle on the transformation comparing to other tools in the market.

Due to the fact that all the data obtained from the design diagram must be imported into the tool and preprocessed, classified, and saved before doing any of the analysis, we must have a database to serve as a repository. Any commercial database software like MS Access, SQL server or Oracle would be sufficient. MS Access is chosen to be the repository of ARAT for the quick development process and convenience. But we prefer to choose Oracle 9i in production environment for the reason of system robustness and performance. The Figure 2 is the process flow chart for this project.

The current version of ARAT is built on Window XP system. This tool is composed by using Java programming language and J2SDK1.4.2 compiler [15]. Any platform

with Java Virtual Machine 1.2 or higher can run this tool. If an appropriate JVM is not available, a binary version can be converted from Java source code. In addition, we integrate commercial package EspressChart 5.0 [14] into the ARAT GUI for intuitive data display.

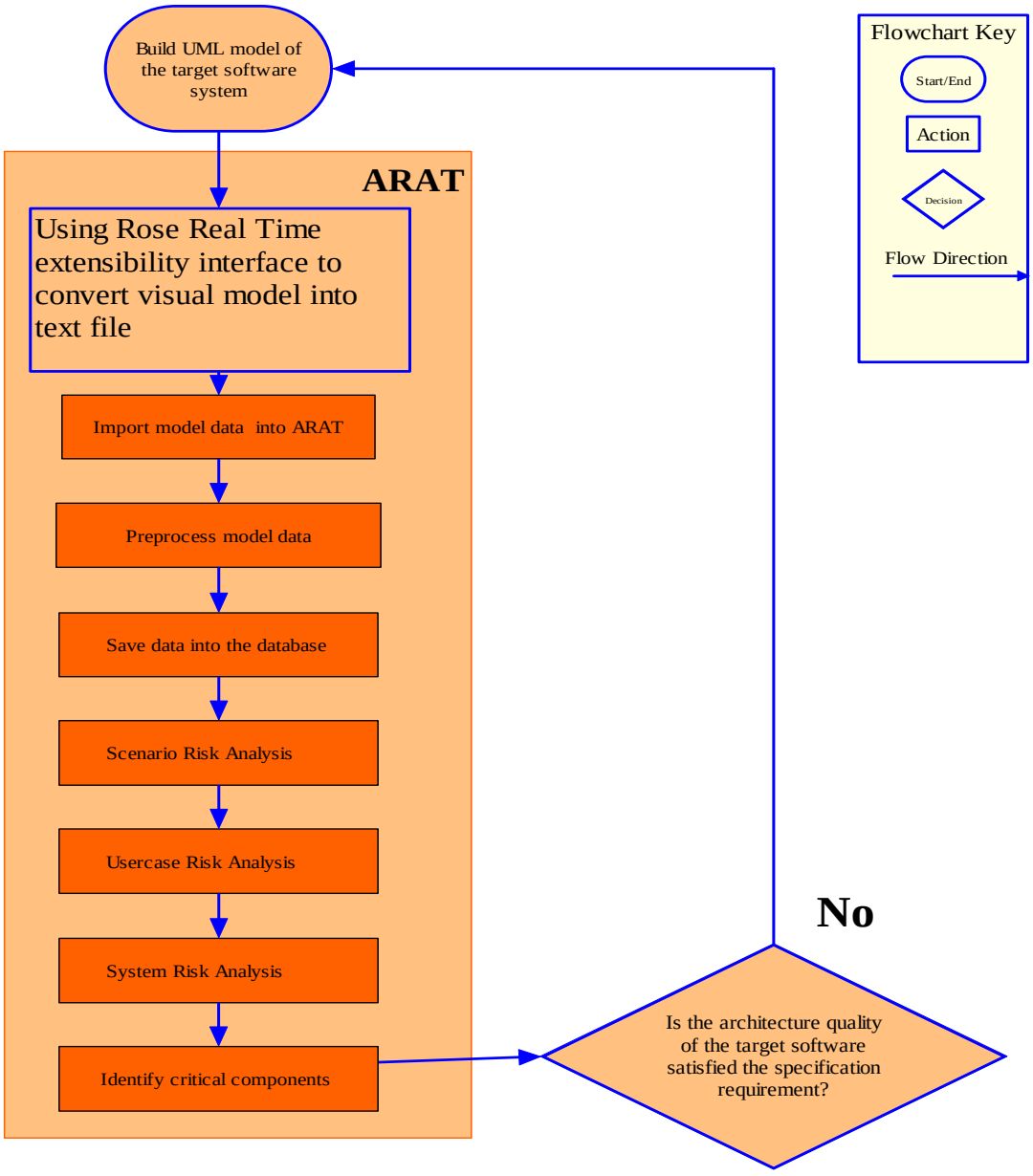


Figure 2: Overall Process Flow chart of ARAT

3.2 Overall system requirements

The system is required to compute the scenario, use case and the overall system risk factors based on our methodology and eventually needs to identify all the critical components in the system. Based on this requirement, ARAT divides all the functions into different calculation modules. Each module is an independent calculation unit. For example, for the complexity calculation module, it first retrieves needed data which has been previously saved in the database. Then, based on the methodology presented in the last chapter, it goes through all the components of every scenario one by one to determine the complexity for every component. Finally, it passes the results to the display module of ARAT and presents it to the user in multiple formats. Meanwhile, it also sends all of its results back to the database for further process which will be carried out by other modules. The Figure 3 is the control flow chart of this module. The overall control flow chart is attached in this thesis as Appendix B.

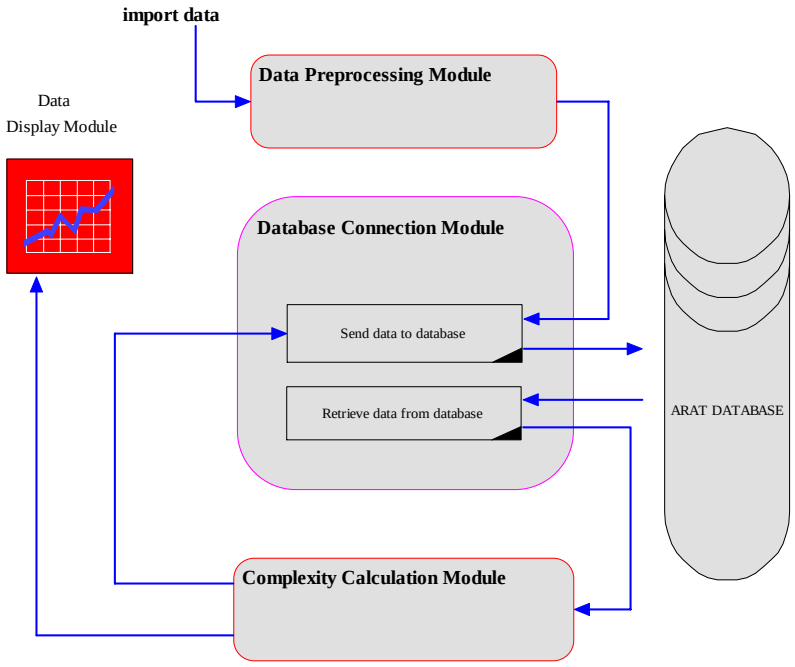


Figure 3: Complexity Calculation Module control Flow Chart

Since all the functions of the system requirement follow the same pattern except some of the modules which need more user interactions, this structure make ARAT system highly componentized. Different modules independently carry out different functions to satisfy their own specific requirements. New modules can be easily added for additional functions in the future with little modification. It even can be simply hooked up by adding new menus or buttons on the display module console GUI after the completion of the development of new module. Figure 4 is one of the examples of console GUI from the ARAT system with many different menu items for different functions or calculation modules.

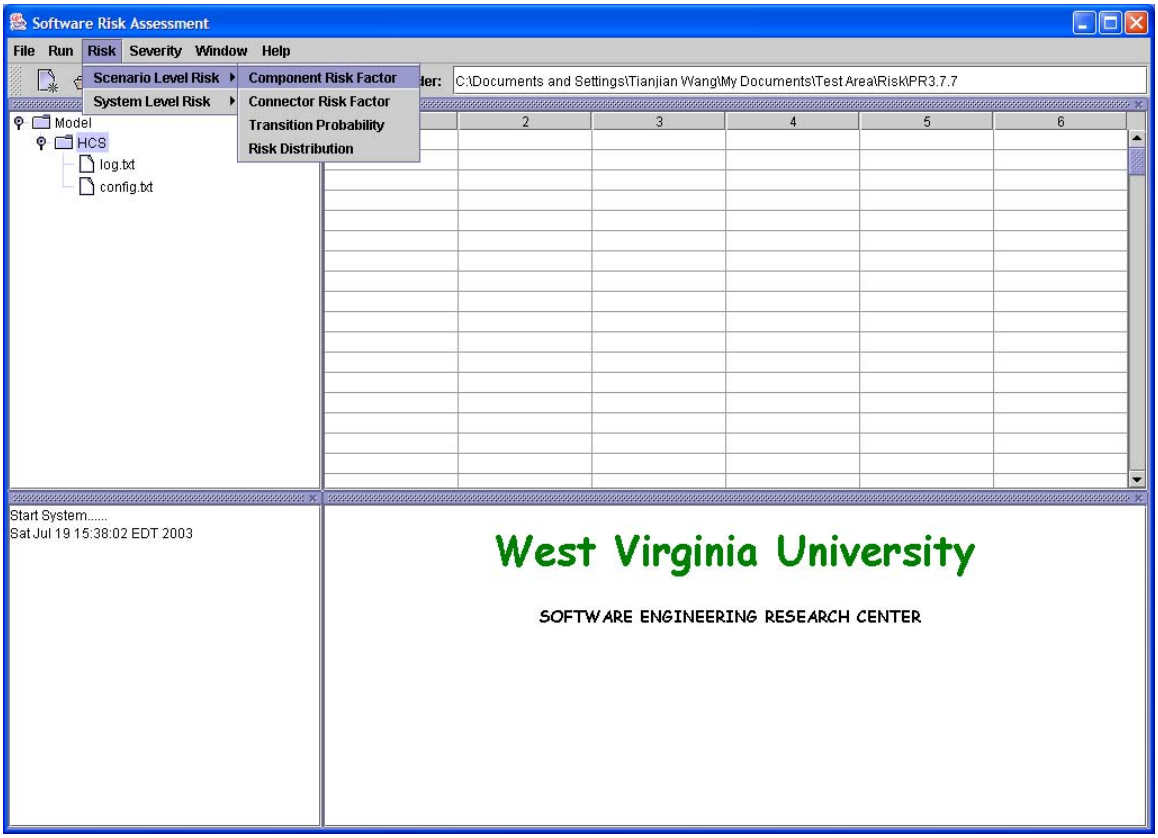


Figure 4: The console GUI of ARAT system

However, based on our methodology to estimate the risk factor heuristically, this system must exactly follow the same way defined in the methodology to carry on

the calculation. That means the ARAT has to calculate the coupling and complexity at different levels before it can move to the next stage for the calculation of risk factors. This makes the module which calculates the risk distribution depend on the completion of those two modules. Since no user interaction is required for coupling and complexity modules, we may make the risk distribution estimation module integrate with them at the background to take off this limitation in case users want to get the results quickly and directly instead of complete, systematically calculation results in detail, this will be discussed in the next chapter.

3.3 User interface requirements

The user interface must follow the system requirements to carry on the calculation. In addition, it must be easy and convenient for user interaction such as obtaining user input, displaying calculation results clearly and precisely, and for good exception handling. The user interface also adds in some system administration functions like target software system administration, user action log etc. Thus, ARAT becomes more powerful to handle multiple tasks simultaneously and keep all the records of user activities.

3.4 Hardware and software requirements

There is no special requirement for the hardware. Any desktop with common or average configuration in the market could be used to run ARAT, For instance, 1.0-2.0 GHZ PIII CPU with 256 Mb Memory, 20 GB hard disk etc. The ARAT system must be combined with a modeling tool to use, we choose Rational Rose Real Time [7] for the build-in interface to obtain the model data and high performance for the transforming process.

CHAPTER 4. DESIGN

This chapter describes the overall architectural design of ARAT and the design of its main components in detail.

4.1 Structure of ARAT

After the user finishes building the UML visual model, the model normally contains use case diagrams and sequence diagrams. Since the diagram is not easily quantified and analyzed to transform the model from diagram information into other format becomes necessary. A textual format is chosen to act as the transient carrier to temporarily save and express the model information. Other formats may also be options, for example XML format with more compatibility and transforming flexibility could be considered as a future development option and will be discussed in the final chapter. ARAT begins with opening the textual data file to import the model data. Some model data which are redundant and need to be filtered out before saving them into ARAT database. A preprocessing module to filter unnecessary data and save the results must be created. As a result, the rest of the calculation modules can systematically retrieve the clean and useful data from the database to do various calculations based on the process control flow. Meanwhile, the module saves the new results generated from the current module back to the database after completing the calculation of new tasks such that it can be used by the next module or simply for the quick access of the next operation when the user requests the same action on the same module data.

The Figure 5 is the use case diagram of the ARAT system. There are mainly 6 use cases determined for ARAT so far. They include most of the actions that would be taken by users. The first use case is for the analyst to collect model information from the UML model. After retrieving useful model information from the depository, the analyst can estimate the dynamic metrics and component/connector risk

factor. Within the procedure of process, the assistance from a domain expert to provide probability data and different weight configuration options for some calculation module is critical to obtain the results. Finally, the high risk components of the target software system can be identified and presented to the analyst on the GUI.

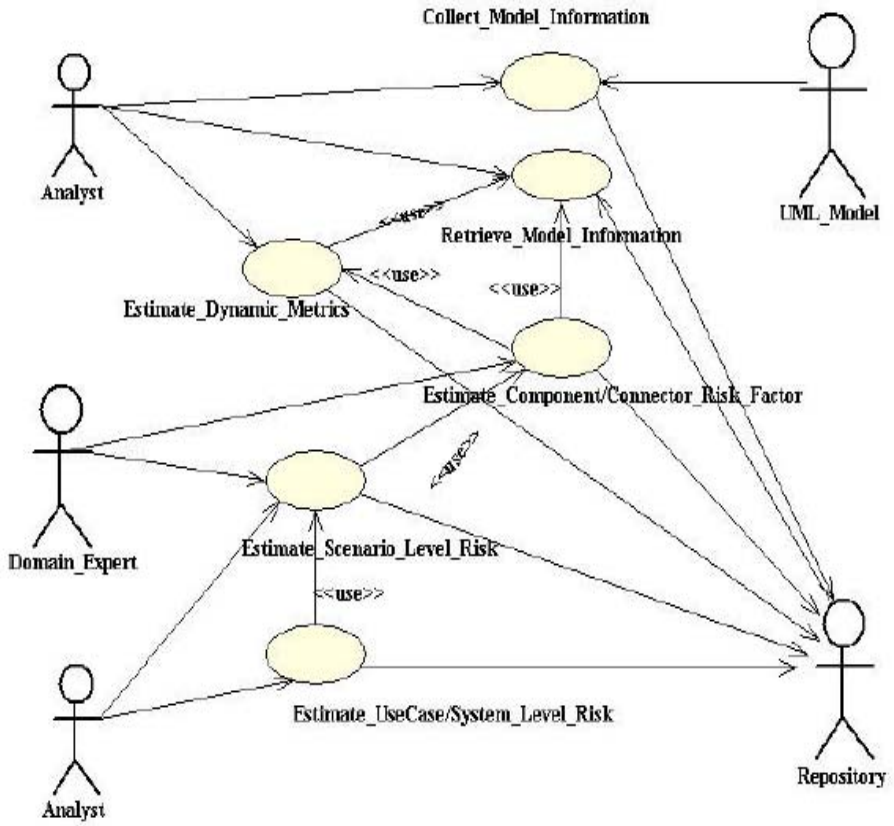


Figure 5: Use Case Diagram of ARAT

Before moving forward to the implementation directly, more detailed design works have been done as well. More design diagrams are created including a component diagram (Figure 6) and a class diagram (Figure 7) based on the UML specification. From the component diagram, we can see that the Quadbase commercial 3D chart package is used by one of the internal Frame which makes up the consol

GUI of ARAT. Even some dynamic libraries from j2sdk are also placed into the diagram like AWTEventMulticastor class. Inside the ARAT package, we have two more special packages. One is for image icons used by the menu Bar of ARAT console GUI. The other one is used to keep instructional html files which will be used by one of the internal frames as well.

The class diagram gives more detail on the implementations of ARAT. Especially for the ModuleCalculation class, most of the implementations of the methodology will be carried out in this class. Hence, most of the efforts and testing will be allocated into this class. InternalFrameSet class is inherited by 4 internalFrame. Each one has its unique property and functions. LogFrame and ModelFrame are created for the purpose of ARAT system management and monitoring. But the TableFrame class and ChartFrame are mainly for the data presentation. This will be discussed in detail at section 4.4.

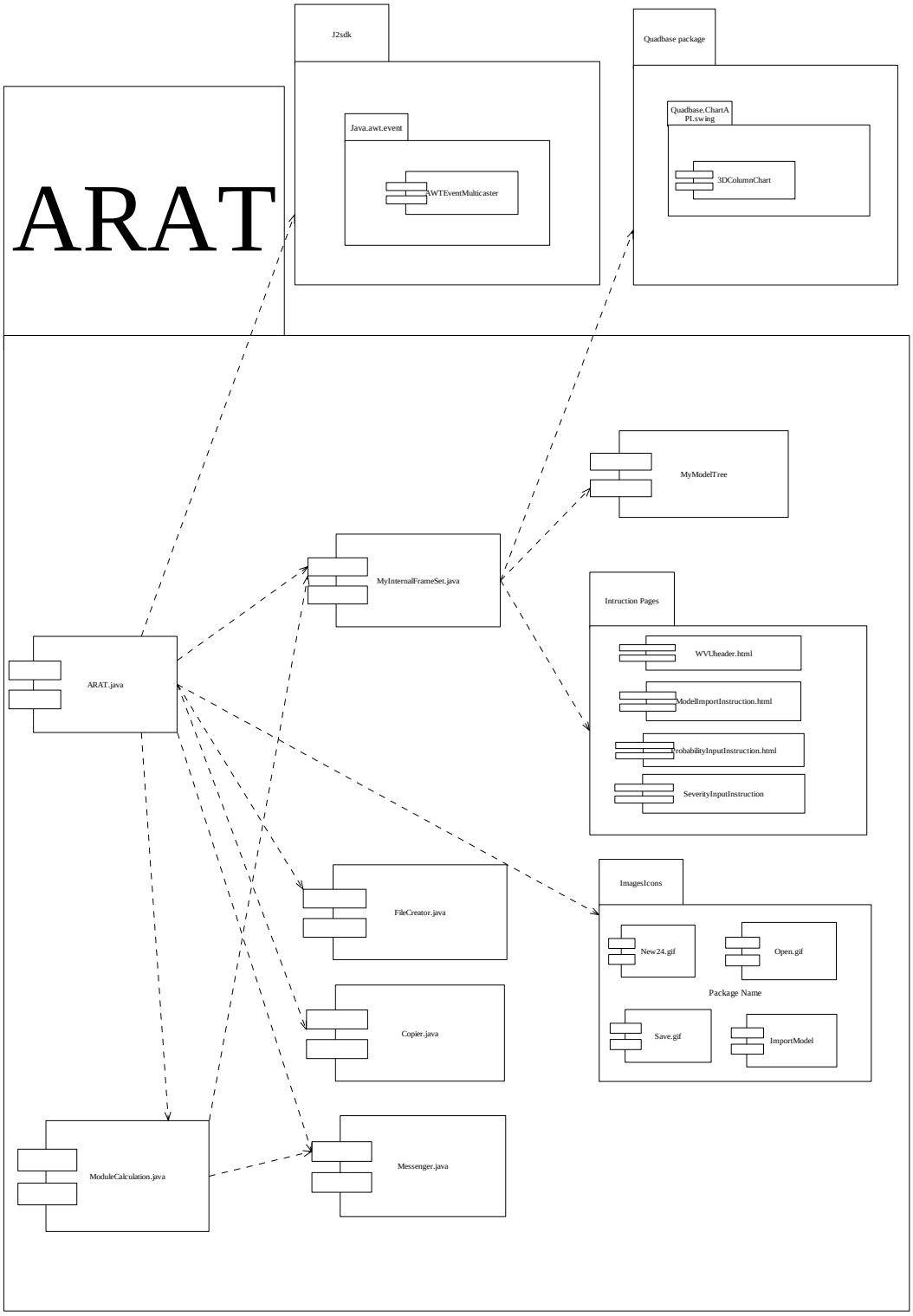


Figure 6: Component diagram for ARAT

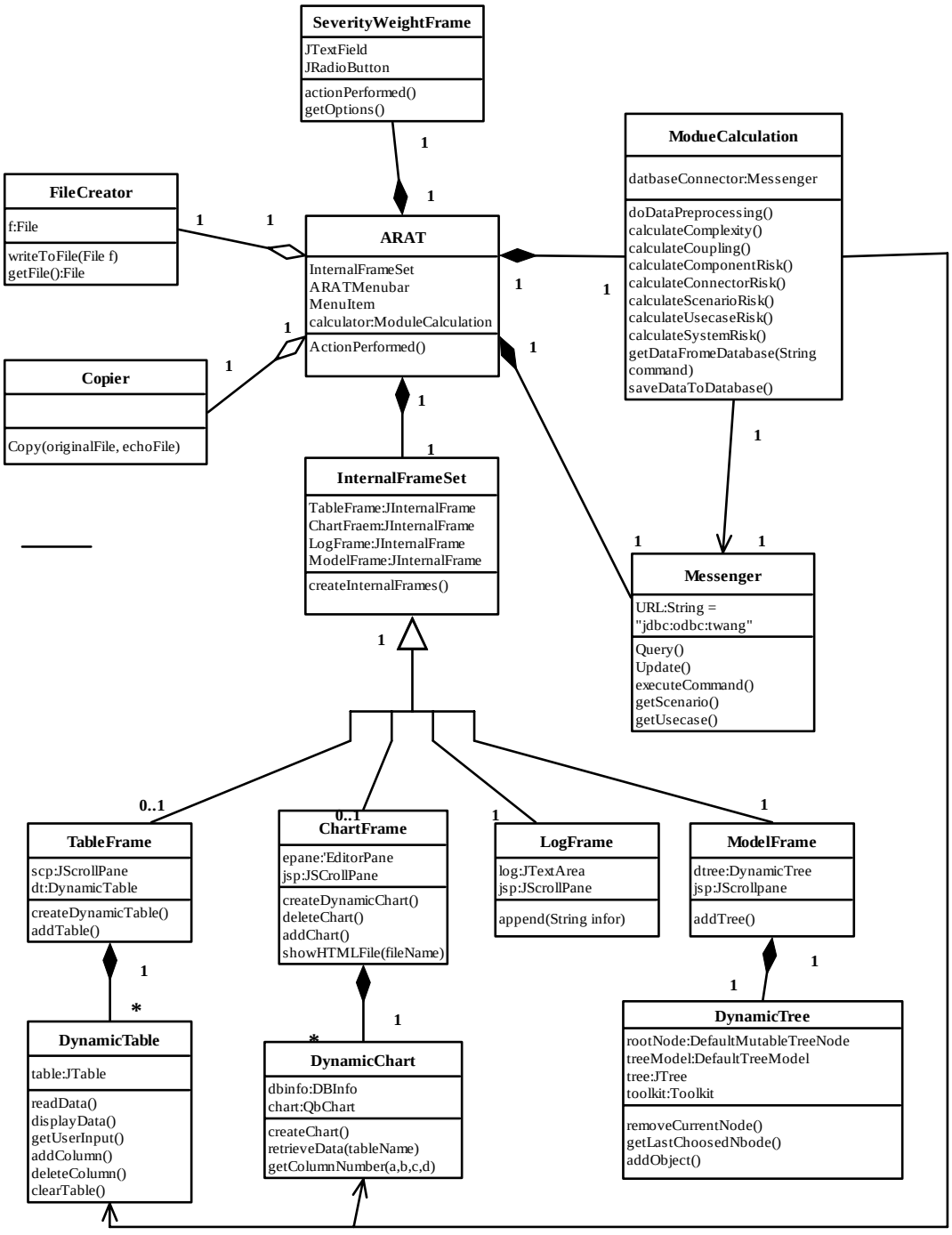


Figure 7: Class diagram of ARAT

4.2 Database Design

Since the development on this project is unfinished, some new modules with new functionalities will be added into ARAT in the next phase of our research. Simultaneously, more extensive data will be imported into the ARAT database as well. The attributes of these data might be totally different from the data used currently; even the method used to analyze these data is still unknown. In order to prevent a large amount of modifications on the database design during the process of adding new modules, ARAT tries to keep maximum flexibility during the design. Thus, it might be able to tolerate minor additions of the functionalities. Otherwise, it simply creates a new cluster of tables while keeping the current tables untouched.

The connection between the database and ARAT use JDBC-ODBC Bridge. All the SQL commands used to create tables and manipulate data are integrated into each calculation module. All the connections between database and the ARAT application are built by Messenger class in which all the parameters are set up.

The ER diagram for the ARAT database is show in Figure 8. We can see that a table named Raw_data is created to keep all the newly imported data distinguished by model, use case, and scenario. Two tables named Component and Connector respectively are used to save data after filtering useless data from Raw_data table. Both tables have reserved space to contain more calculation results from another calculation module. For examples, the attributes risk_factor is used to hold the component/connector risk calculation result for each component and connector. But it will not be filled up when the calculation process is at Complexity/Coupling calculation phase. The severity attribute is used to hold the severity entered form ARAT GUI by users only when the process comes to the Component/Connector risk factor calculation phase.

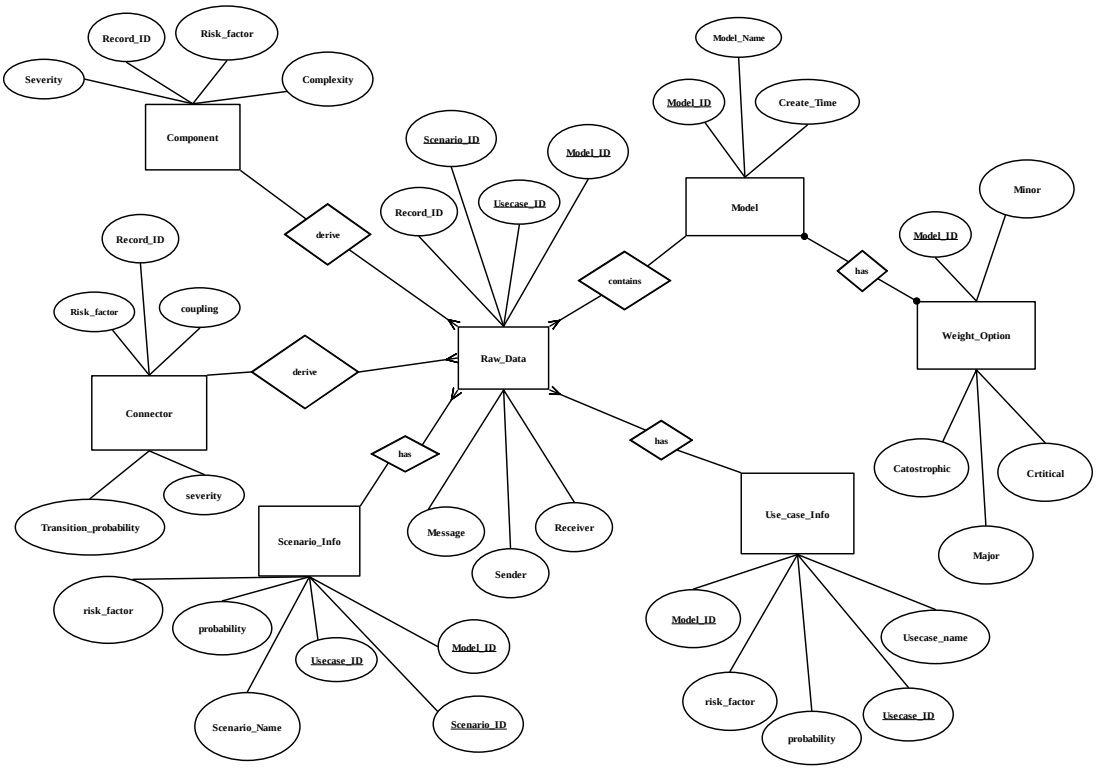


Figure 8: ER diagram for ARAT database

4.3 Calculation Module Design

Two most primitive calculation modules are for complexity and coupling. Both modules retrieve data from the Component and Connector tables when user actions are captured from the console GUI. After the calculations are finished, each module sends the result back to database and simultaneously sends data to the GUI module for presentation, how the GUI module presents these data will be discussed in detail in the next section.

ARAT also includes other calculation modules. Like transition probability module, scenario risk factor module, use case risk factor module, scenario risk distribution module, markov calculation module, overall system risk module etc, all the

modules have the same pattern to retrieve data and save data. The procedure is summarized as follows:

Step 1: retrieve existing data from tables. If data does not exist, give warning instructions to the user.
Step 2: carry out the calculations based on methodology.
Step 3: save results to the table for other calculation modules.
Step 4: send data to the GUI module for presentation.

4.4 GUI Module Design

4.4.1 Presentation Component Design

This module acts as the interface between ARAT and users. It gets parameter/commands provided by the user and presents the information/results back to the user. For example, the GUI component receives user commands to calculate dynamic complexity and sends this command to the database to request data needed to do the calculation, then passes those data to the dynamic complexity calculation module. After finishing the calculation, the GUI module receives the result back from the calculation module and displays it to the user. The interface design normally requires data to be presented clearly and precisely, in addition, the critical component of the target software system must be distinguishable. So a presentation component is necessary, for data display, one way is to keep numerical data in a table, the other way is to use graphical presentation. ARAT integrates them together for maximum readability and clarity. Both tabular format and 3D charts are used to build up the presentation component. The tabular Frame displays the results data directly in order to keep data precision. Figure 6 is a tabular Frame embedded in ARAT GUI. It's expandable and up to the maximum size of the GUI frame for best visibility and clarity of data as well as ease and convenience of user input.

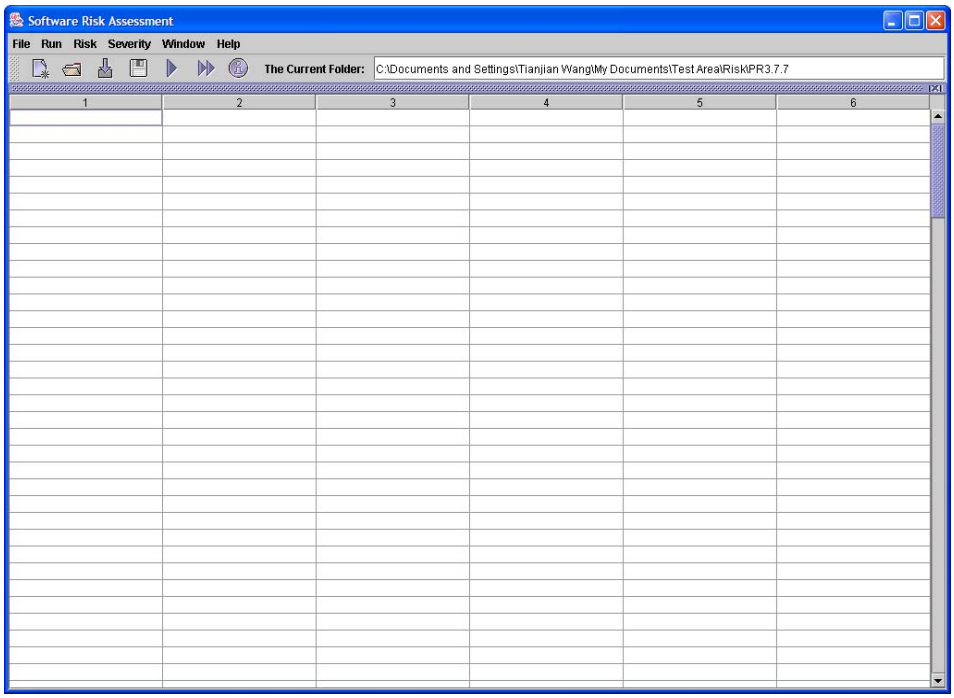


Figure 9: Maximized Tabular Frame

Simultaneously, a graphical component can display all the calculation results/data in 3D chart format. This can let the user very easily identify the most critical components. For example, the bar column in the chart below, Figure 7 maximized 3D chart frame, indicates the risk factor of every component under each scenario and use case. Whenever the user points the mouse arrow to the column, the chart automatically displays a small yellow popup menu to show the name of the component, scenario, use case as well as the value of the column. On the other hand, it can help a user quickly and easily find the column which represents critical component/connector/scenario/use case etc with highest values.

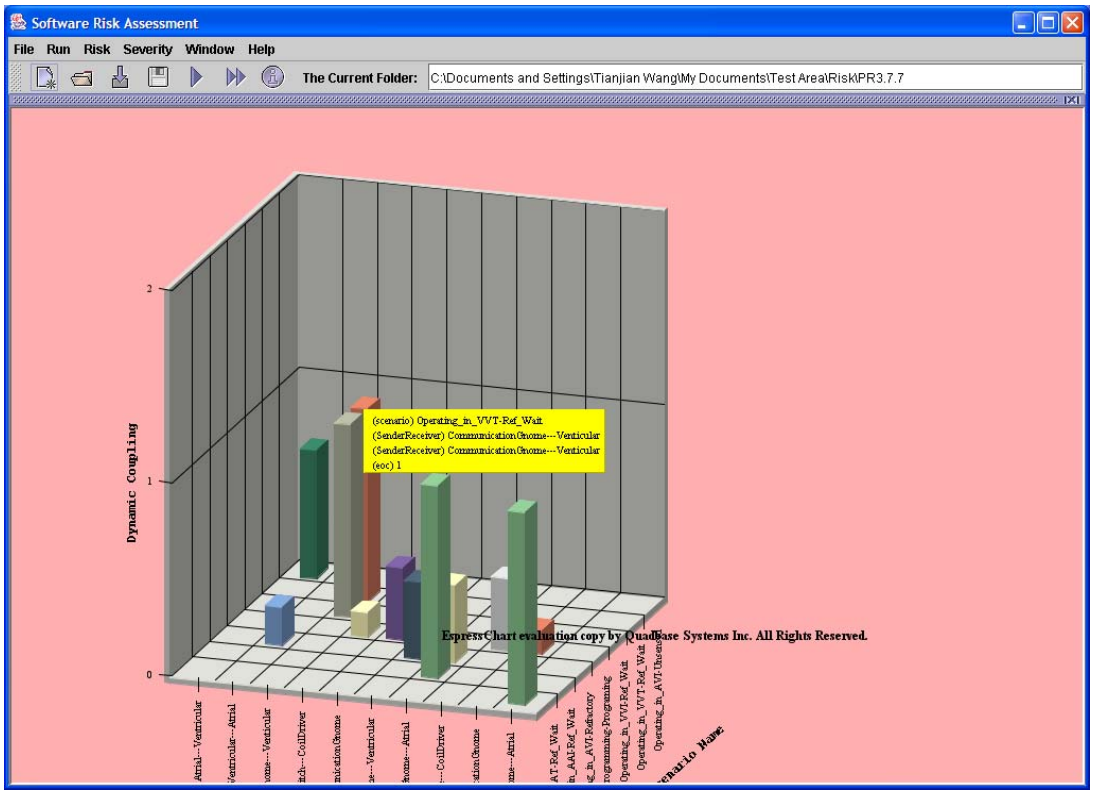
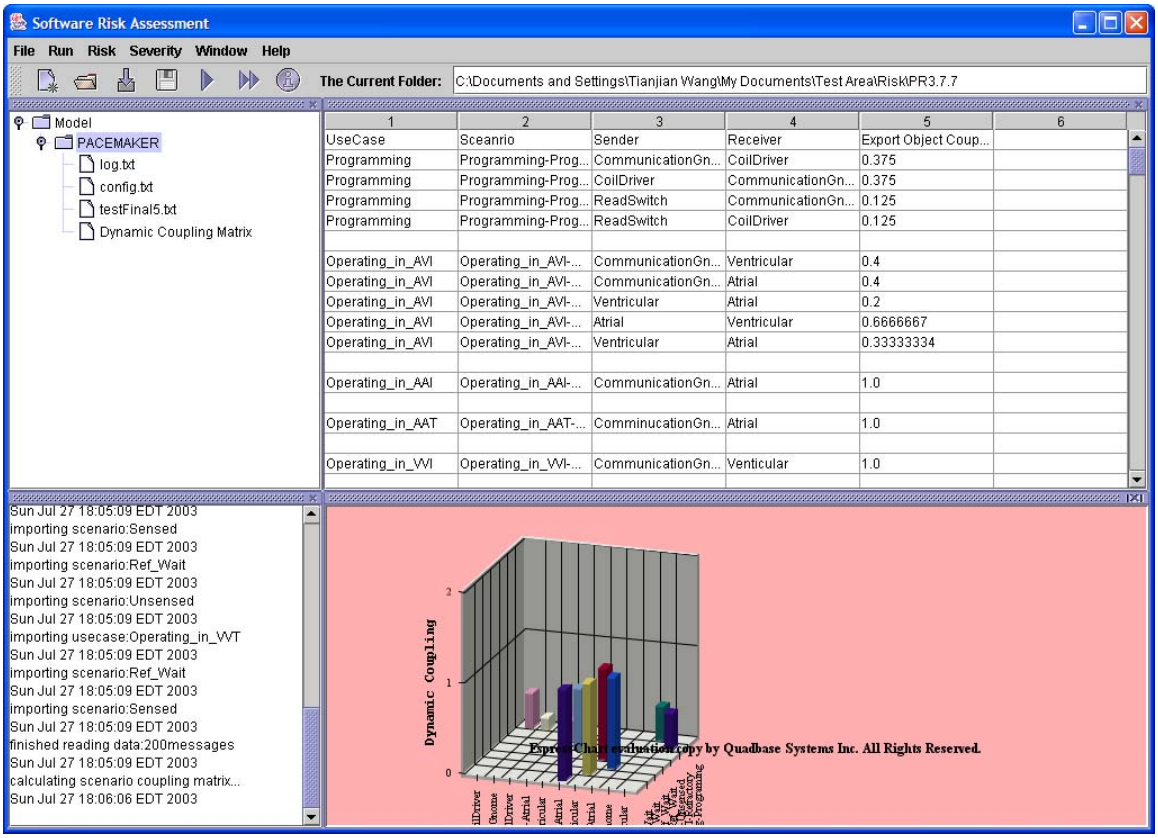


Figure 10: Maximized 3D Chart Frame

Both Figure 9 and Figure 10 are single frames in maximized expanding status for maximum visualization and clarity. They are integrated together with other GUI components. When the user needs the system overview of comprehensive information, Figure 11 is an example; it provides maximum information of target system and the status of ARAT to the user. The left upper frame provides information for target software systems management, for instance, ARAT can analyze many different software systems simultaneously. This is especially useful when the user wants to compare multiple solutions of UML models for the same target system. The left lower frame is a log window; it records every user action applied to the ARAT system, in case some accidental actions happen, the user can follow the log to reverse the user action, and erase the error made by the accident.



1	2	3	4	5	6
UseCase	Sceanrio	Sender	Receiver	Export Object Coup...	
Programming	Programming-Prog...	CommunicationGn...	CoilDriver	0.375	
Programming	Programming-Prog...	CoilDriver	CommunicationGn...	0.375	
Programming	Programming-Prog...	ReadSwitch	CommunicationGn...	0.125	
Programming	Programming-Prog...	ReadSwitch	CoilDriver	0.125	
Operating_in_AVI	Operating_in_AVI-...	CommunicationGn...	Ventricular	0.4	
Operating_in_AVI	Operating_in_AVI-...	CommunicationGn...	Atrial	0.4	
Operating_in_AVI	Operating_in_AVI-...	Ventricular	Atrial	0.2	
Operating_in_AVI	Operating_in_AVI-...	Atrial	Ventricular	0.6666667	
Operating_in_AVI	Operating_in_AVI-...	Ventricular	Atrial	0.33333334	
Operating_in_AAI	Operating_in_AAI-...	CommunicationGn...	Atrial	1.0	
Operating_in_AAT	Operating_in_AAT-...	CommInucationGn...	Atrial	1.0	
Operating_in_VVI	Operating_in_VVI-...	CommunicationGn...	Venticular	1.0	

Figure 11: GUI component Overview

4.4.2 Interactive Component Design

In the ARAT system, a lot of information needs to be entered through user interactions. Such as the severity estimation for every connector/component, it needs to be entered by the user manually the first time the user applies the target system to ARAT. Based on Shneiderman's (1998) classification, there are five primary styles for user interaction, ARAT chooses two of them: Form fill-in and Menu selection. Some advantages are obvious in terms of the objectives of ARAT, Such as simple data entry and error avoidance.

Following Figure 12 is an example of Form fill-in design, Three user options regarding the severity weight configuration are shown on the frame, the user could either choose the default severity scale option or input data to define their own severity weight scale from the text field .

Figure 12: Severity Weight Option Frame

There are some other user interaction GUI components, including table input and dialog input. These components will be explained in detail in Chapter 5.

4.5 Extensibility and Compatibility

Based on the progress of our software engineering research, ARAT must have extendibility to integrate more functional modules. Obviously, an interface to hookup these modules must be reserved. This can be easily done by adding more

menu items on the consol GUI, at the same time, adding separate calculation functions into ModuleCalculation class. In addition, adding extra cluster of tables for new modules is feasible.

CHAPTER 5 IMPLEMENTATION

This chapter describes the implementation process of ARAT system including most of the modules. It also talks about how to make use of the Rose RT extensibility interface, how to create the database, and how to integrate the commercial package EspressChart[14] into ARAT GUI module.

The Incremental Development Process [16] is adopted for ARAT implementation. This process can ensure the low risk of overall project failure[12]. We first design each calculation module as an increment. Implementation and verification of each increment are conducted independently, such that, the overall failure is not likely to happen even if one of the modules fails to meet the requirements. By using this software development process, the risk of ARAT system failure is significantly reduced.

5.1 Development Environment

The development environment is the open source software Eclipse 2.1.1 from IBM Corp. under the terms and conditions of the Common Public License 1.0. Using Eclipse IDE has some obvious advantages, such as automatic syntax check, automatic brace generation, parentheses and quote check, etc. All class member popup, multiple perspective and views make management and development more convenient and easy. Many plug-in interfaces make Eclipse become more powerful than any other Java IDE, for example, the UML plug-in from Omondo company can easily import diagrams even the whole package of the model from many model tools directly, it also can provide an interface for a user to directly create diagrams in Eclipse, afterwards, the frame of the source can be generated based on the diagrams or packages provided by user. Figure 13 is a screenshot of the Eclipse IDE platform.

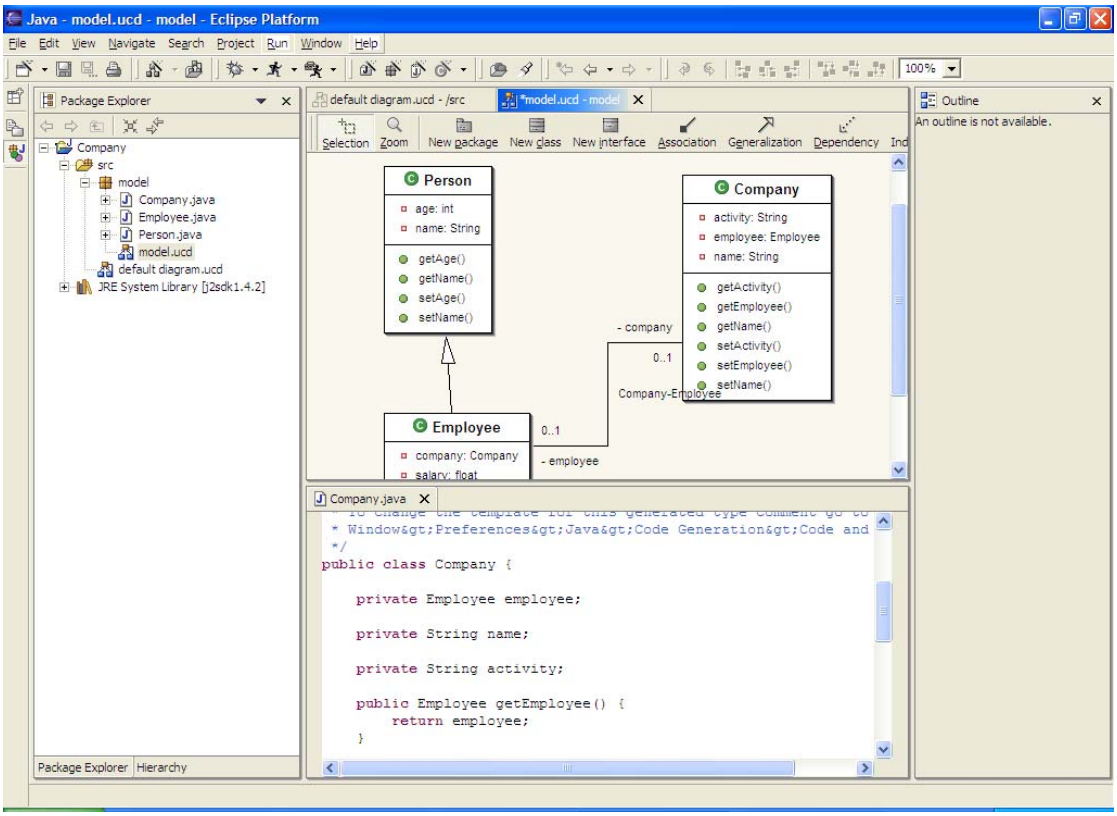


Figure 13: Eclipse IDE platform

J2sdk1.4.2 [15] is the newest version of Java compiler and JVM from Sun Corp. Since ARAT need a lot of GUI components to express the analysis of data, using the newest version JVM and compiler can make full use of the highly object oriented java classes such that user defined GUI components can be created by simple and efficient source code. Meanwhile, significantly using Swing classes in javax package makes the ARAT GUI module much easier to be controlled than using AWT classes. In addition, the new *Java HotSpot Server VM* [15] replaces the classic JVM obtaining significant performance enhancement, the GUI repaint can be 4 times faster than before, this is achieved by using an adaptive compiler which could analyze the source code and detect performance bottlenecks, or "hot spots". JVM compiler compiles those performance-critical portions of the code for

a boost in performance, while avoiding unnecessary compilation of seldom-used code (most of the program), then the adaptive compiler decides on the fly, how best to optimize compiled code. Finally, it provides rapid memory allocation for objects, is a fast, efficient, state-of-the-art garbage collector. All these advantages make the ARAT GUI module gain very high computation speed; the high performance even surprised the personnel while I was doing the demo at the NASA annual meeting.

5.2 Instruction of Rose Real Time Extensibility Interface

Rational Rose is a well established UML modeling tool in software engineering. Rose RT is specifically useful for real time system design. Especially, Rose RT can extend and customize its capabilities to meet the specific software development needs. It has the following main capabilities.

1 Customize Rational Rose Real Time menus

2 Automate manual Rational Rose Real Time functions with Rational Rose Real Time Scripts (for example, diagram and class creation, model updates, document generation, etc.)

3 Execute Rational Rose Real Time functions from within another application by using the Rational Rose Real Time Automation object.

4 Access Rational Rose Real Time classes, properties and methods right within your software development environment by including the Rational Rose Real Time Extensibility Type Library in your environment.

In order to satisfy the requirement of ARAT and simplify the conversion process of transforming visual data into quantitative data, we take advantage of the second feature. We compose a rose real time script at the beginning of the risk

assessment process. The script automatically goes through all the diagrams of the target software UML model. Specifically, it acquires the relationship information between different use cases from use case diagrams, meanwhile, detail information between components and connectors are acquired from sequence diagrams as well.

The following data (Figure 14) which describe the relationship between different use cases is captured from use case diagram of the UML model displayed in Figure 15.

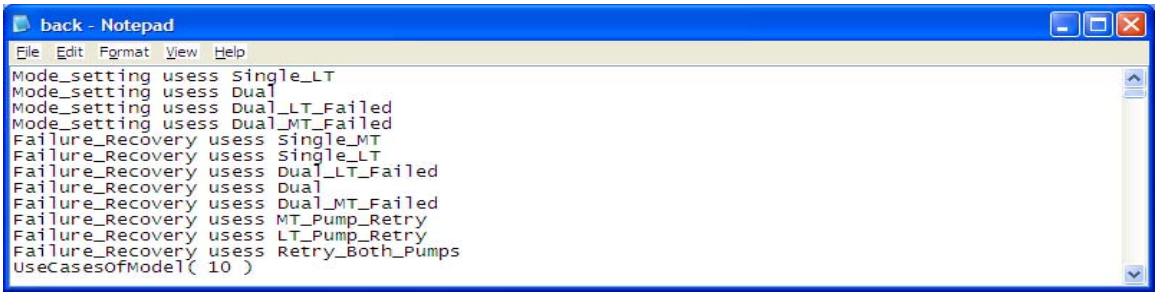


Figure 14 Information captured from use case diagram

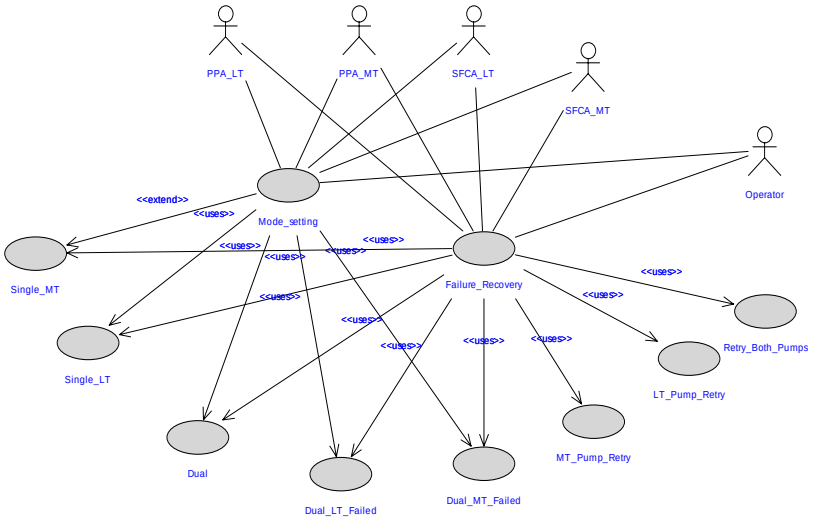


Figure 15 Example use case diagram of target software system

The part of rose script being used to capture the information from use case diagram is as follow:

```
Set assocs = RoseApp.CurrentModel.GetAllCategories.GetFirst("UseCase View").Associations
    Dim anAssoc As Association
    For i% = 1 To assocs.Count
        Set anAssoc = assocs.GetAt(i)

        If anAssoc.stereotype = "uses" Or anAssoc.stereotype = "extend" Then
            If anAssoc.Role1.Class Is Nothing Then
                r1Name$ = anAssoc.Role1.UseCase.name
            Else
                r1Name$ = anAssoc.Role1.Class.name
            End If

            If anAssoc.Role2.Class Is Nothing Then
                r2Name$ = anAssoc.Role2.UseCase.name
            Else
                r2Name$ = anAssoc.Role2.Class.name
            End If

            If anAssoc.Role1.navigable = TRUE Then
                If anAssoc.stereotype = "extend" Then
                    Print #1, r1Name$ & " " & anAssoc.stereotype & "s " & r2Name$
                Else
                    Print #1, r2Name$ & " " & anAssoc.stereotype & "s " & r1Name$
                End If
            Else
                If anAssoc.stereotype = "extend" Then
                    Print #1,r2Name$ & " " & anAssoc.stereotype & "s " & r1Name$
                Else
                    Print #1,r1Name$ & " " & anAssoc.stereotype & "s " & r2Name$
                End If
```

Figure 16 rational rose script module for use case diagram

The same way is used to scan the sequence diagram of the UML model and convert it to textual data (Figure 17). The following is the partial results after going through one of the sequence diagram (Figure 18 is a part of sequence diagram) in a scenario by running the Rose RT script.

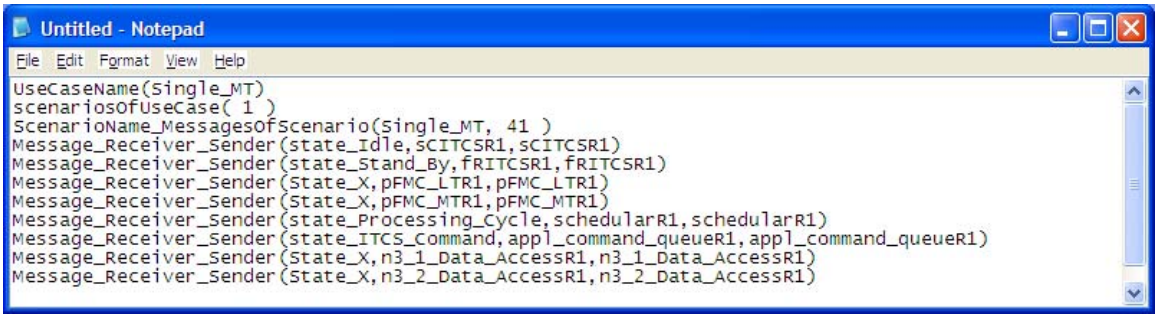


Figure 17: textual data presentation of sequence diagram

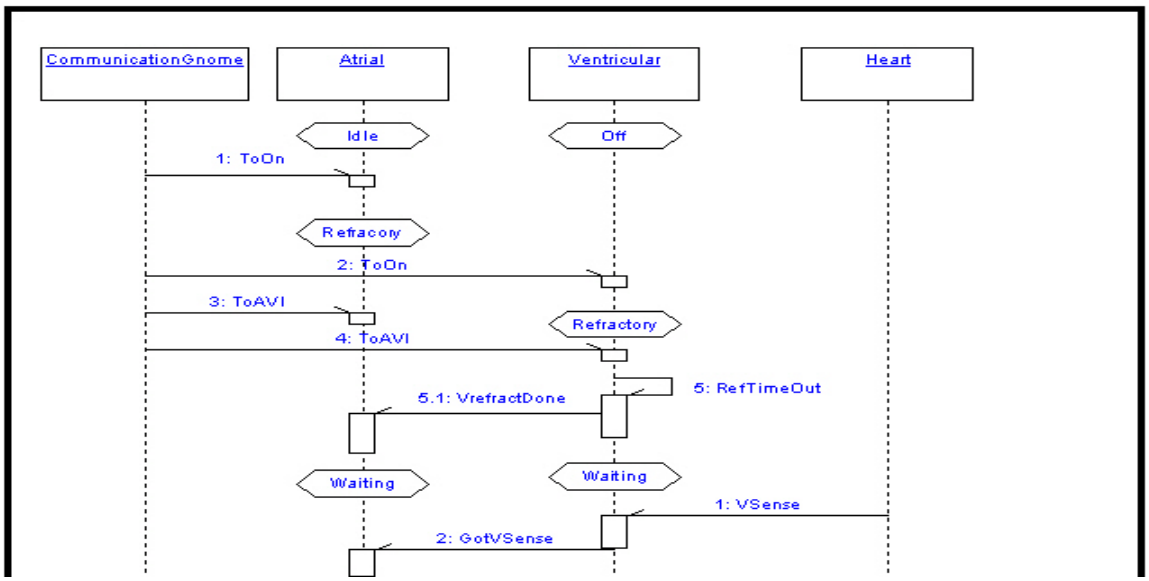


Figure 18 Sequence Diagram Example

The part of rose script module used to convert the sequence diagram into textual data is as follows:

```
For j%=1  To  ScenarioDiagrams.Count
      set theScenario = ScenarioDiagrams.GetAt(j)
      Set theMessages = theScenario.GetMessages ( )
          temp$ = left$ (theScenario.Name , 14)
          If  StrComp(temp$, "Collaboration1")   Then
          Print
#1,"ScenarioName_MessagesOfScenario(";theScenario.Name;",";theMessages.c
ount;")"
          Else
          End If
 For k%=1 To theMessages.count
      Set theMessage = theMessages.GetAt(k)
      Set theObject2=theMessage.GetReceiverObject ( )
      Set theObject3=theMessage.GetSenderObject ( )
   Print #1,
"Message_Receiver_Sender(";theMessage.Name;",";theObject2.Name;",";theObj
ect3.Name;")"
Next k
message_counter=message_counter+theMessages.count
Next j
```

Figure 19: rational rose script module to capture sequence diagram

5.3 Database Implementation

This section will explain ARAT how to set up the connection between ARAT computational component and database, how to create tables from java source code for each module dynamically.

5.3.1 JDBC-ODBC Bridge and SQL Command Handler

In ARAT, a class called Messenger is created to handle all the communication between database and ARAT, inside the class, the connection is actually built on

top of JDBC-ODBC bridge which is the standard connection between java application and MS Access database. The special focus of this connection module is a SQL command handler, which consists of two methods, one is called query(), it can handle all the query SQL command passed from calculation modules,and the other one is called update() which is created in the same way to handle all the update SQL commands. Both query() and update() methods are wrapped up in one method to handle any SQL command from ARAT computational module called executeCommand(). The following is the specification of this method.

```
public LinkedList executeCommand(String cmd) {
//input: any SQL command
//output: the result of the execution of the SQL command
//function: pass all kinds of SQL command to database, return the result.
}
```

5.4 Integrating EspressChart Package

In order to express the data in graphical format for easy identification of critical components, ARAT embedded a commercial package called EspressChart from Quadbase System Inc. this package provides a pure java library with a set of functions to generate different type of advanced two or three-dimensional charts to express the analysis results. The data used to populate charts can be either read from data file (including Excel spreadsheets) or directly come from JDBC bridge which is connected with database.

The configuration for integrating this package into ARAT is quite simple, the user only needs to add the absolute path of the location of this package into the system variable, then the package can be directly imported into the source code. The software API is also included in the package for convenient reference.

5.5 GUI

From the discussion of the graphical presentation of data in section 4.4, GUI module design, both tabular format and 3D charts are chosen to express data. Each of them are placed inside an internal frame, please refer to Figure 9 and Figure 10 for detail, However, the console GUI (Figure 11) not only consists of tables and charts, it also includes a Toolbar to hold some image icons which are some shortcuts for receiving commands from users. A Text Field to indicate the current working directory is embedded into the Toolbar. In addition, another two internal Frames are created to be responsible for software model administration and user action log. Multiple window configurations on the GUI have some advantages which allow different information to be displayed simultaneously on the screen. The user can switch from frame to frame (or window to window) without losing sight of information in another frames. On the other hand, the disadvantage for this configuration is obvious that the context or data in each frame might not be clear enough to be read even though scrollbars are provided due to each frame only occupying part of the screen, so the capability of expanding each frame to full screen size become necessary, however, user preference at any moment is the only consideration.

.

CHAPTER 6 TESTING

6.1 Functionality testing and integration testing

Upon finished implementation of each single calculation module of ARAT and the completion of the whole ARAT system, extensive testing is carried on to decide if it's satisfied our project objectives. The testing include functionality testing on individual calculation module(or called validation testing) and integration testing when each calculation module is integrated into ARAT system . User interface is tested at the final stage.

Since the development process of ARAT use an incremental model, each calculation module is designed as increment and is implemented separately, before a new module is integrated to the ARAT system, the module must pass the functionality testing or increment verification testing. The functionality is tested on multiple case studies include the Pacemaker model which is an example of a critical real-time application: A cardiac pacemaker is an implanted device that assists cardiac functions when the underlying pathologies make the intrinsic heartbeats low. An error in the software operation of the device can cause loss of a patient's life. We first build the UML model in Rational Rose Real Time, then based on our methodology, a simulation of calculation method is implemented by using Matlab[17] build-in functions. Finally the calculation result of each simulation is compared with the the result from ARAT calculation module. For example, the Markov chain calculation is embedded into each scenario risk calculation, it's implemented as a single calculation module with different data input from various scenarios among many different use cases. The mathematic calculation is complicated and errors are likely to occur during the programming process, it's even impossible to do the calculation manually, fortunately, the Matlab has some built-in functions that can easily complete the task. If a very simple simulation of the calculation is built, it only take several lines of code to implement it in the

Matlab environment. Since the result is more likely to be correct comparing to the result from ARAT implementation, most of the verification testing of the ARAT module is conducted in this way. Only when the result from ARAT module implementation is verified by our testing model Pacemaker and several other case study models provided by our research group, this module is ready to integrate into the ARAT system.

For the integration testing, a top-down approach is used to determine the compatibility of each computational module and the system integrity. The main functions of the ARAT tool is tested by applying the same case study used on module verification. Even if a couple of case studies might not be enough, it still is expected to expose some latent defects before applying massive testing. Furthermore, in each module, each function is tested. Some problems can be easily detected and fixed, but some are not. For example, after integrating the scenario risk distribution into the system, the size of the 3D chart located inside one of the Internal Frame objects does not expand and shrink accordingly while the user might want to change the size of the Frame based on his personal preference. This problem does not occur when only testing the GUI component independently, so a further action is taken to specifically test this risk distribution module, and finally some modification has to be made to the GUI module.

6.2 User interface testing

Dealing with massive data is a big hassle and error-prone; working on a friendly user interface can reduce the accidents caused by user actions. The user interface testing is carried out by different people including common CSEE students in our department and most of the research group members of this project. Each person is introduced briefly to this tool, and the process of risk assessment. All of them can quickly understand how to use the ARAT tool, how to convert the UML model into textual data, and import the data into the ARAT tool, however, some minor

modifications are made to the GUI module to make the data presentation more clear. For instance, the labels of some 3D charts previously have been overlooked, although the legend and title are clearly provided for each chart or even if the user knows what process he is focusing on, but it might still cause some confusion when multiple charts with plenty of data are presented simultaneously.

ARAT is expected to be applied on more case study models in the future. In the current phase of our research, the testing result is satisfied and the objectives of the project are met.

CHAPTER 7 ANALYSIS AND CONCLUSION

7.1 Analysis and conclusion

This project explores the risk assessment tool ARAT. It is entirely based on the methodology proposed by our research group. ARAT estimates the distribution of the scenario/use case/system risk factors on different severity classes which allow us make a list of critical scenarios in each use case, as well as a list of critical use cases in the system. Finally, we identify a list of critical components and connectors that have high risk levels in high severity classes. The results could guide the allocation of development and testing effort based on critical use cases, scenarios, components, and connectors. The results from this stage of work are satisfied and have shown the feasibility of implementing complex risk assessment processing algorithms in ARAT.

7.2 Future work

This work could extend to other software engineering analyses where all the analyses need to be conducted at the early phase during the lifecycle of the software system.

For the future implementation, ARAT can integrate new calculation models for hazard analysis to allow automatic and precise estimation of the severity level for each architectural element. ARAT is also planned to integrate new models to calculate all the static metrics in the target software system even though the results maybe not as sensitive and complete as dynamic metrics for early risk assessment.

REFERENCE

[1] N. Fenton, N. Ohlsson, “Quantitative Analysis of Faults and Failures in a Complex Software System”, IEEE Trans. Software Engineering, Vol. 26, No. 8, pp. 797 -814, 2000.

[2] W. Harrison, “Using Software Metrics to Allocate Testing Resources”, Journal of Management Information Systems, Vol. 4, No. 4, 1988, pp. 93-105.

[3] K. Goseva-Popstojanova , A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, H. Ammar, A. Mili, “Architectural-Level Risk Analysis using UML”,IEEE Transactions on software Engineering, Vol.29, No.10, Oct 2003

[4] J. Rumbaugh, I. Jacobson, G. Booach, The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.

[5] A. Hassan, W. M. Abdelmoez, R. M. Elnaggar, H. H. Ammar, “An Approach to Measure the Quality of Software Designs from UML Specifications,” 7th International Conference Information Systems, Analysis and Synthesis, 2001, Vol.IV, pp.559-564.

[6] NASA Technical Std. NASA-STD-8719.13A, Software Safety, 1997.

[7] UML Language Resource Center: Unified Modeling anguage, Standard Software Notation, http://www.rational.com.

[8] M. Stojanovic, K. El-Emam, “ES1: A tool for collecting objectoriented design metrics”, *NRC/ERB-1087*, May 2001.

[9] M. Hitz, K. Neuhold, “A Framework for Product Analysis”, OOPSLA *1998 Workshop on Model Engineering*, Methods and Tools Interaction with CDIF, 1998.

[10] L. Nenonen, J. Gustafsson, J. Paakki A. Inkeri Verkamo, “Measuring object - oriented software architectures from UML diagrams”, Proc. *4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2000, pp. 87-100.

[11] NASA Safety Manual NPG 8715.3, Jan. 2000.

[12] Ian Sommerville, Software Engineering, 6th edition , Addison-Wesley, 2000

[13] A. Hassan, W. Abdelmoez, R. Elnaggar, and H. Ammar, “An Approach to Measure the Quality of Software Designs from UML Specifications,” Proc. Fifth World Multi-Conf. Systems, Cybernetics and Informatics, vol. 4, pp. 559-564, July 2001.

[14]EspressChart 5.0 evaluation copy, Quadbase systems Inc.

[15] Java 2 standard development kit 1.4.2 , http://www.sun.com

[16] Mills, H.D., O’Nell, D. et al., The management of software engineering.IBM Sys. J.,24(2), 414-77 1980

[17]Matlab version 6.5, The MathWorks, Inc. http://www.mathwork.com

[18]Rational Rose RealTime, vesion 2002.05.00 Rational Software Corporation, http://www.rational.com

[19] T.Wang, A. Hassan, A. Guedem, W. Abdelmoez, K. Goseva-Popstojanova, H. Ammar,” Architectural Level Risk Assessment Tool Based on UML Specifications”, the 25th International Conference on Software Engineering ICSE.2003

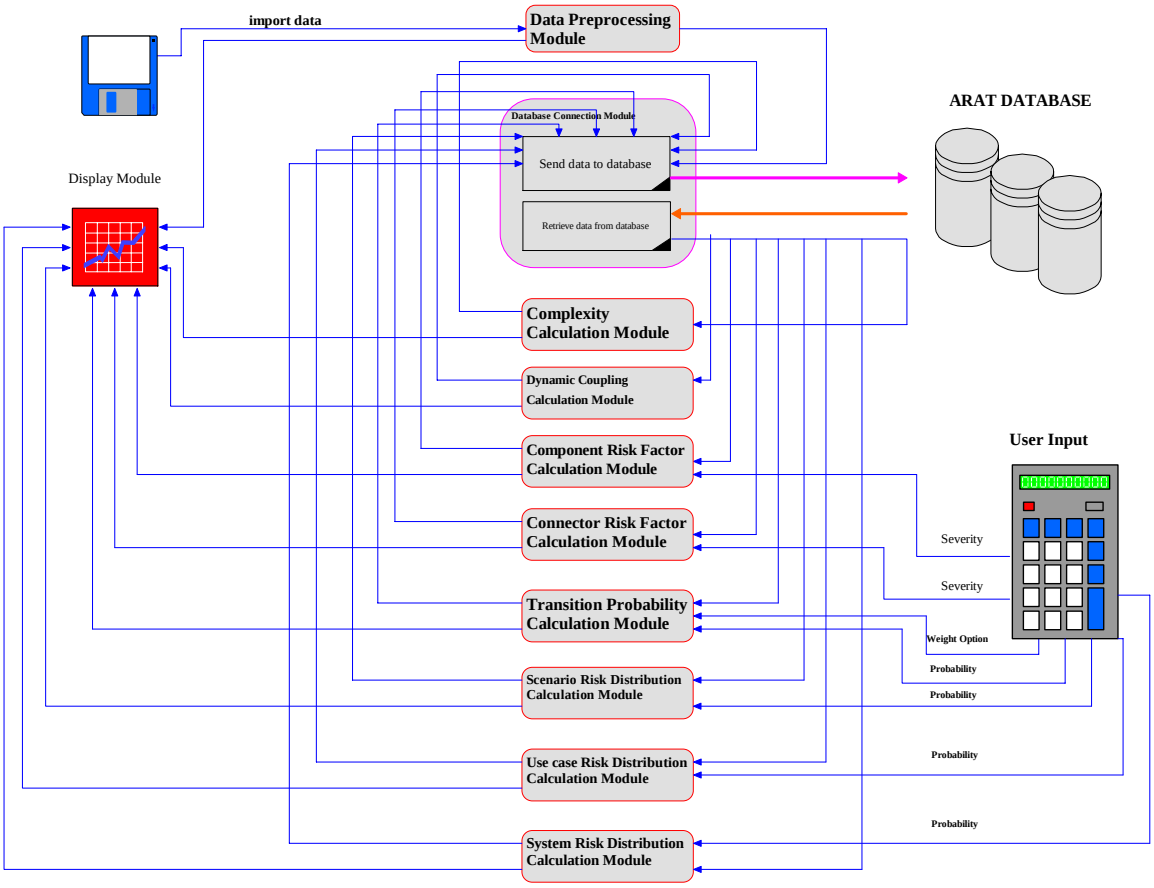
APPENDIX A Rose Real Time Script for model conversion

```
Sub writeModelDocumentation(FileName  As  String)

Dim temp$
Dim AllUseCases  As UseCaseCollection
Dim theUseCase   As UseCase
Dim theModel     As Model
Dim theScenario  As Scenariodiagram
Dim theMessages  As MessageCollection
Dim theMessage   As Message
Dim totalMessage As Integer
Dim message_counter As Integer

Dim theObjects As ObjectInstanceCollection
Dim theObject  As ObjectInstance
Dim theObject2 As ObjectInstance
Dim theObject3 As ObjectInstance

Dim ScenarioDiagrams As ScenarioDiagramCollection
set theModel = RoseApp.CurrentModel
Set Categos =theModel.GetAllCAtegories
Set AllUseCases=theModel.GetallUseCases

Dim assocs As AssociationCollection

Open FileName$  For  Output  Access  Write  As  #1

    Set assocs = RoseApp.CurrentModel.GetAllCategories.GetFirst("Use Case View").Associations
    Dim anAssoc As Association

    For i% = 1 To assocs.Count
        Set anAssoc = assocs.GetAt(i)
```

```
        If anAssoc.stereotype = "uses" Or anAssoc.stereotype = "extend" Then
            If anAssoc.Role1.Class Is Nothing Then
                r1Name$ = anAssoc.Role1.UseCase.name
            Else
                r1Name$ = anAssoc.Role1.Class.name
            End If

            If anAssoc.Role2.Class Is Nothing Then
                r2Name$ = anAssoc.Role2.UseCase.name
            Else
                r2Name$ = anAssoc.Role2.Class.name
            End If

            If anAssoc.Role1.navigable = TRUE Then
                If anAssoc.stereotype = "extend" Then
                    Print #1, r1Name$ & " " & anAssoc.stereotype & "s " & r2Name$
                Else
                    Print #1, r2Name$ & " " & anAssoc.stereotype & "s " & r1Name$
                End If
            Else
                If anAssoc.stereotype = "extend" Then
                    Print #1,r2Name$ & " " & anAssoc.stereotype & "s " & r1Name$
                Else
                    Print #1,r1Name$ & " " & anAssoc.stereotype & "s " & r2Name$
                End If
            End If
        End If
    Next i

Print #1, "UseCasesOfModel(";AllUseCases.count;")"
   message_counter=0
   totalMessage = 0

For i%=1 To AllUseCases.count
   Set theUseCase = AllUseCases.GetAt(i)
   Set ScenarioDiagrams=theUseCase.ScenarioDiagrams
```

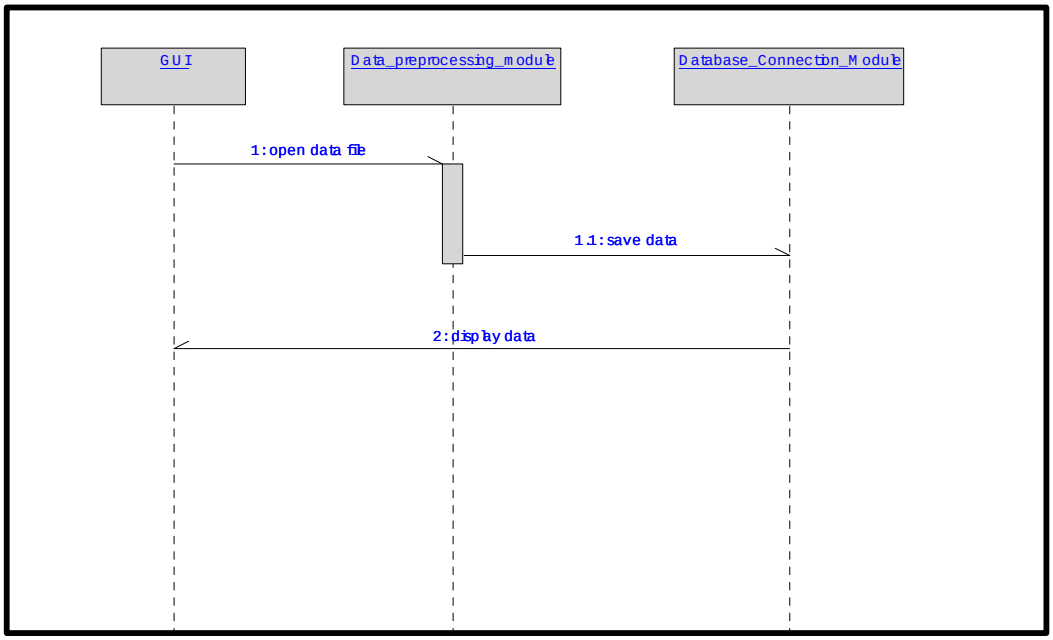
```
   Print #1,"UseCaseName(";theUseCase.Name;")"
if ScenarioDiagrams.Count > 0  then
   Print #1, "scenariosOfUseCase(";ScenarioDiagrams.Count-1;")"
End If
For j%=1  To  ScenarioDiagrams.Count
        Set theScenario = ScenarioDiagrams.GetAt(j)
        Set theMessages = theScenario.GetMessages ( )
        temp$ = left$ (theScenario.Name , 14)
        If  StrComp(temp$, "Collaboration1")   Then
            Print                                                                    #1,
"ScenarioName_MessagesOfScenario(";theScenario.Name;",";theMessages.count;")"
        Else
End If
For k%=1 To theMessages.count
        Set theMessage = theMessages.GetAt(k)
        Set theObject2=theMessage.GetReceiverObject ( )
        Set theObject3=theMessage.GetSenderObject ( )
Print #1, "Message_Receiver_Sender(";theMessage.Name;",";theObject2.Name;",";theObject3.Name;")"
Next k
         message_counter=message_counter+theMessages.count
Next j
         totalMessage = totalMessage + message_counter
        Print #1,"MessagesOfUseCase(";message_counter;")"
        message_counter=0
 Next i
        Print #1,"MessagesOfModel(";totalMessage;")"
End Sub

Sub Main

      FileName$=SaveFileName$("writeModelDocumentation","Text files:*.txt")
      If  FileName$<> "" Then  writeModelDocumentation FileName$
 End Sub
```

APPENDIX B ARAT overall control flow chart

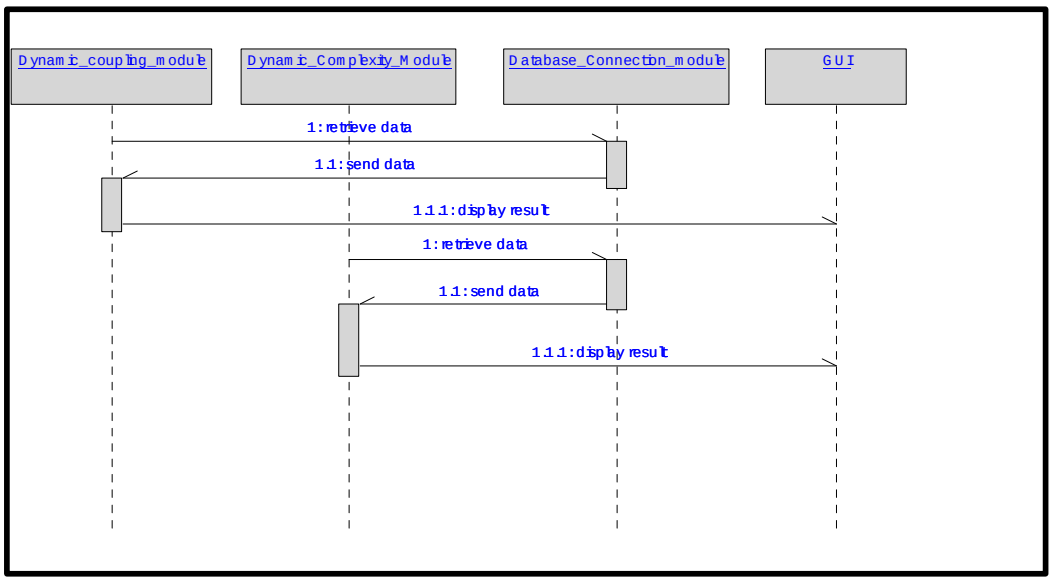


APPENDIX C ARAT Sequence Diagra

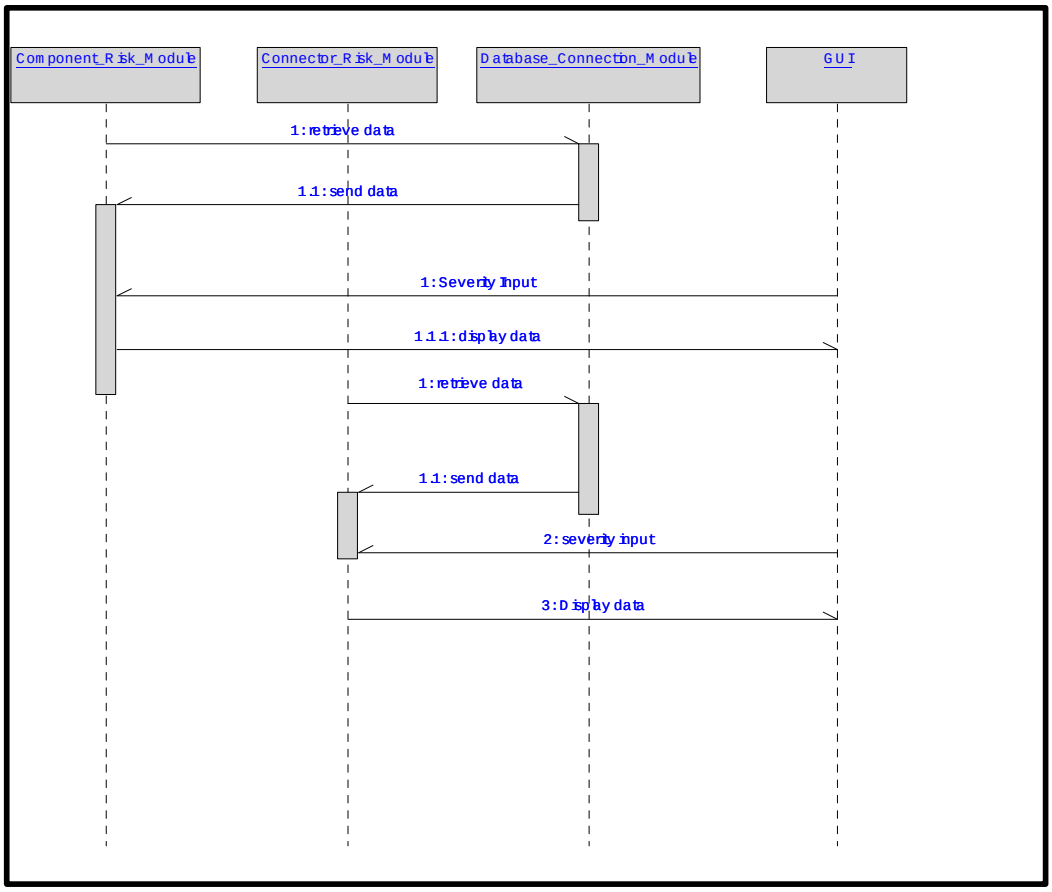
Sequence diagram for Retrieve_Model_Infor scenario



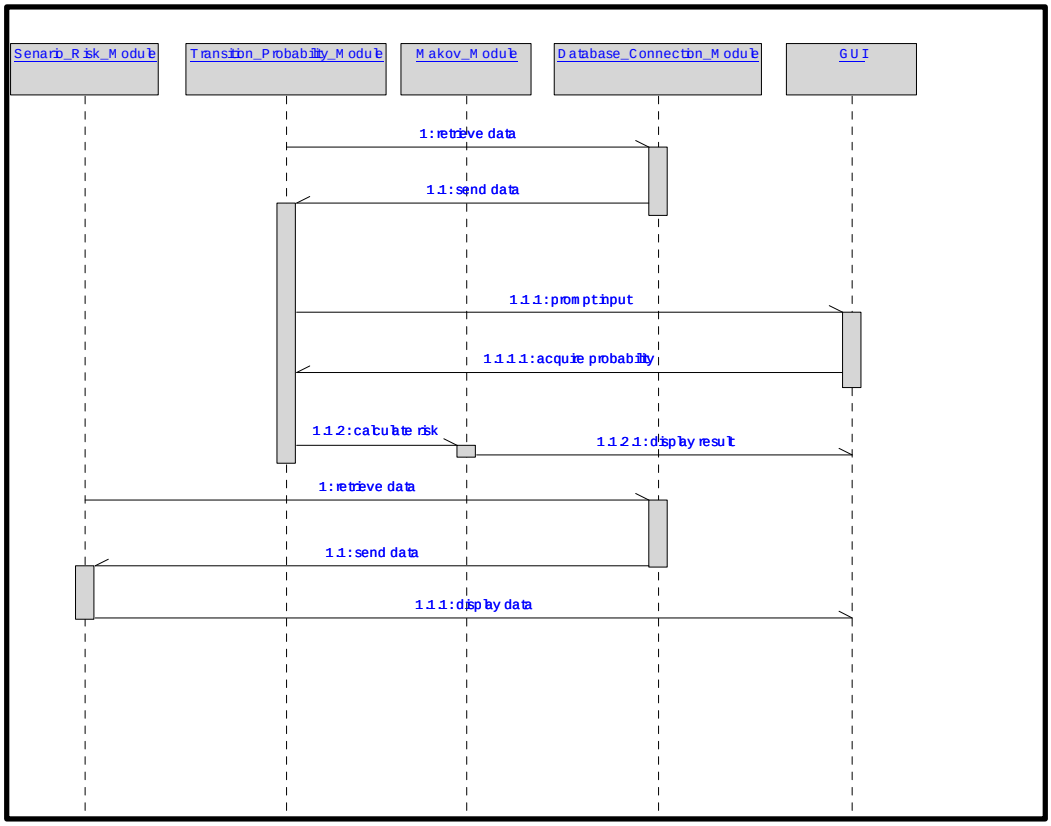
Sequence diagram for Estimate_Dynamic_Metrics scenario



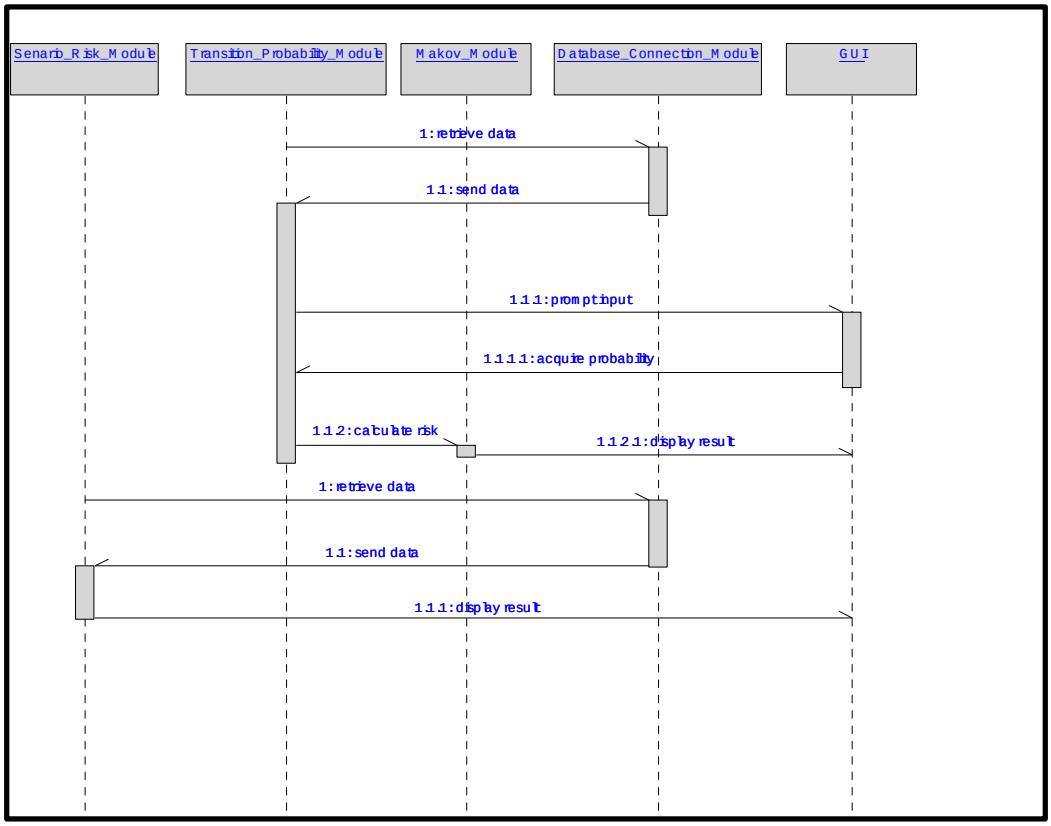
Sequence diagram for Estimate_Component/Connector_Risk scenario



Sequence diagram for Estimate_Scenario_Risk Scenario



Sequence diagram for Estimate_Usecase_Risk_Module scenario



Sequence diagram for Estimate_System_Risk scenario