CHALMERS |  UNIVERSITY OF GOTHENBURG

Visualizing cyber attacks with misuse case maps

Master of Science Thesis in Software Engineering and Management

YASHAR BIZHANZADEH

Supervisor : Gerardo Schneider
Co-supervisor: Peter Karpati, Guttorm Sindre

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, November 2013

Abstract

Business processes require a supporting technical architecture enabling their realization. The secure design of this architecture has key importance in the future of the business; however there are not many notations which consider security issues and architectures together. Misuse case map is a diagrammatical notation that supports the depiction of system architecture, together with security vulnerabilities and their mitigations. It also enables the representation of intrusion scenarios through exploit paths. The diagrams can become quickly complex with the arising details of the system and the intrusion, thus tool support is a big step forward for spreading the use of these notations. This master thesis introduces jMUCMNav; an editor for misuse case maps based on the well tested and widely used use case map editor.

Keywords: security requirements, misuse case maps, system architecture, editor

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank people who helped me during this master thesis which would have been impossible without their support and advice.

Gerardo Schneider - Associate Professor in Chalmers University of technology who was an academic supervisor of mine. I would like to thank him for his efforts and devotion towards me throughout my thesis work and two years study at IT University of Gothenburg-Chalmers.

Peter Karpati - Post Doctoral Fellow in Norwegian University of Science and Technology who was an academic co-supervisor of mine. I am extremely thankful for his extreme effort, guidance and advice to push me forward during my study.

Guttorm Sindre - Professor in Norwegian University of Science and Technology.

Arash Bizhanzadeh - Java Architect in CBC-Canada. Lastly, I offer my best regards and blessings to my lovely brother who supports me technically and mentally during my master thesis.

Table Of Contents

1. Introduction

Security is a qualitative, anti-functional or non-functional requirement in software system development. It is considered in the software development life cycle right from the early requirement analysis phase. With the growing importance of secure systems, security requirement engineering (SRE) grows as a part of requirements engineering (RE). This new field feeds into the RE by introducing different approaches to cope with security issues. Different approaches in SRE looks at security related problems from different point of views. For example, we can mention multilateral approach, UML based approach and misuse case maps (MUCM) in this area. They provide different ways towards the goal of defining security requirement for their target system. In what follows, we will take a look at their different routes toward designing a security critical system.

First, the multilateral approach reaches its goal by looking at the security issues in different viewpoints. It considers different security expectations of stakeholders as well as the conflicts among these expectations to define the security requirements of its target system. Next, the UML based approach utilizes unified modeling languages to define the security requirements. Finally, misuse case maps (MUCM) make a relation between the architectural component and security requirements. It is a diagrammatical modeling technique which connects security related considerations to the technical architecture of an enterprise system, thus providing a reliable base for a secure architecture.

One of the major problems this area faces today is supporting these approaches and techniques with additional tools to increase efficiency and usability. Some well-known editors and workbenches in the SRE field, like Surakasha, SQUARE and SeaMonster have been developed to support different approaches and techniques. Unfortunately, none of these workbenches support the MUCM modeling technique, which makes it difficult to use MUCM in a complicated industrial project.

The aim of this thesis work is developing an extendible editor tool for MUCMs**.** The main concern is developing this editor tool followed by designing and running an experiment to evaluate its efficiency and usability. We will answer the following research question during this study

- **Is it feasible to develop an extendible, efficient and usable editor for MUCM**

This thesis is organized as follows: we start by describing our research methodology in the next section. Then we provide the background required to understand the main concepts in this thesis work. Then we use a case study on different security modeling editors to reveal the best design and architectural solutions for developing our tool jMUCMNav. Finally, an experiment is designed to assess the efficiency and usability of the editor.

2. Research Methods

In this section, the main objectives and research questions of this master thesis are explained. We present our research in different steps and we explain which methods are used in each step to find answers for the main research questions.

Answering to the following three questions leads us to the answer of the main research question.

- Should we develop a MUCM editor based on an existent editor or should we develop it from scratch?
- Is this tool extendible for meeting the future additional requirement?
- Is this tool more efficient than other similar tools?
- Is this tool more usable in comparison with other similar tools?

To answer these questions, the research is divided into three steps. The first step is a background study in the domain of Security Requirement Engineering. The main research method in this step is literature review in the security requirement analysis area.

The second step is a study to find the best way to develop an editor for MUCM modeling language. In this step, a case study is used for finding the best approach for representing MUCMs. Four different editors for similar modeling languages are investigated to find the best design solution. In this step we will answer the first two questions and we will choose to develop it from scratch or use other similar editors as the base of our development.

In the last phase, an experiment with questionnaires is designed to assess the efficiency and usability of the editor. This experiment uses Latin square experiment design and technology acceptance model (TAM). The main methods to collect data during this phase are questionnaires followed by a quantitative data analysis method.

3. Background

This section provides background knowledge and definitions in the SRE field. We will discuss how SRE grows as a part of RE (Requirement Engineering) to make today's reliable and security aware systems. We will go through different approaches in this field and will introduce different techniques related to these approaches. Finally, we will talk about dealing with security issues in the context of architecture, and MUCM modeling language.

3.1. Security requirements engineering

Dealing with security issues after finishing the development, is a common problem even in today's mature software. Security holes unveil after serious system intrusions and consume a big amount of the development budget. These problems turned a special attention toward

SRE. SRE is a part of the requirement engineering process, and starts in the early phase of the software development lifecycle in parallel with RE to define the security goals. Two critical questions are when and how we should start to deal with security in an under developed system. We are going to answer these questions in this section.

"If you don't know what you want, it's hard to do it right" [7]. This statement has a very important role in defining security requirements. Before starting to define the security policy for the system, we should know what we are going to secure, against whom and to what extent. As a best practice, the development life cycle starts with defining functional requirements, the system architecture and the working environment. After that we will be able to make a threat analysis in our special context of functionalities, architecture and environment. Threat analysis is followed by risk analysis to evaluate the severity of each threat and ends with defining security policies for different threats [7].

Now it is almost clear when we are going to start threat analysis to extract security requirement, but it is still unclear how we should extract these security requirements. Should we use the same techniques as we use to extract ordinary functional requirements? To what extent these two types of requirements and their elicitation methods are different? Elicitation methods for ordinary functional requirements usually are not enough for preparing a complete and consistent set of security requirement. Security researchers provide new methods to elicit security requirements in a systematic way. Existing SRE techniques use different approaches and focus on different viewpoints in designing secure software. In the next section we will go further through these methods and techniques.

3.2.SRE techniques and methods

The ultimate goal of SRE is protecting the confidentiality, integrity and availability of the information and resources in its target system. These three key goals are known as the CIA triad in information security area and defined based on the ISO/IEC 13335-1:2004 as follows:

- Integrity is the property of safeguarding the accuracy and completeness of assets.
- Confidentiality is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes.
- Availability is the property of being accessible and usable upon demand by an authorized entity

There are different approaches in SRE to look at security issues from different perspectives, and supported by different techniques and methods. All of them are trying to reach three SRE goals in different ways. In the following sections we will try to look at some of these approaches and their supporting techniques.

3.2.1. Multilateral approach

Multilateral approaches focus on the different security and privacy expectations of the stakeholders and try to solve the conflicts, which always happen between the expectations of these groups. These approaches try to look at the system with the viewpoints of different stakeholders, find out the valuable asset for different group of stakeholders, make the security for all groups, and solve security related conflicts between these groups [6]. These approaches result in multilateral security requirement analysis (MSRA) and some other methods like SQUARE (security quality requirements engineering methodology).

SQUARE is a well-known method in this approach which should be done jointly with requirement engineers and stakeholders. SQUARE focuses on clear communication between stakeholders and security requirement engineers to extract security goals, develop proper artifacts and estimate the security risks by using techniques such as use cases, misuse cases and attack trees. The ultimate goal of these approaches is fully integrating SRE in to the software development process.

3.2.2. UML-based approach

UML-based approaches utilize the notation of the Unified Modeling Language to define security requirements. Misuse cases are a UML-based approach, introduced by Sindre and Opdahl [8] to elicit threats to use cases and functionalities. We will make a comparison between use cases and misuse cases in the next paragraph to make misuse cases more clear.

Use cases that are used in the first steps of the functional requirement analysis, show the functionality of the software and depict the scenarios which users allowed to do with the software. Misuse cases were introduced to show threats to use cases and the functionalities which are not allowed to be performed by users.

For example, in Fig. 1 we represent use cases and misuse cases in an online shopping store. We shows use cases as white circles initiated by users and misuse case as black circles initiated by misusers. “Customer” as a user in this online store can trigger use cases like “Order goods”, and “Outside Crook” as a misuser can trigger misuse cases like “Steal card info” that is a treat for “Order goods” use case.

Lodderstedt et al. present another UML-based approach for distributed systems called SecureUML [9]. It makes a role-based access control for class components using a UML profile to fulfill the confidentiality and integrity goals. “SecureUML can be considered as a notation to specify and design secure software systems, rather than a security requirements engineering method” [7].

UMLsec [10] introduces a UML-based modeling language and focuses on the three core goals of confidentiality, integrity and availability to define the security for security critical systems.

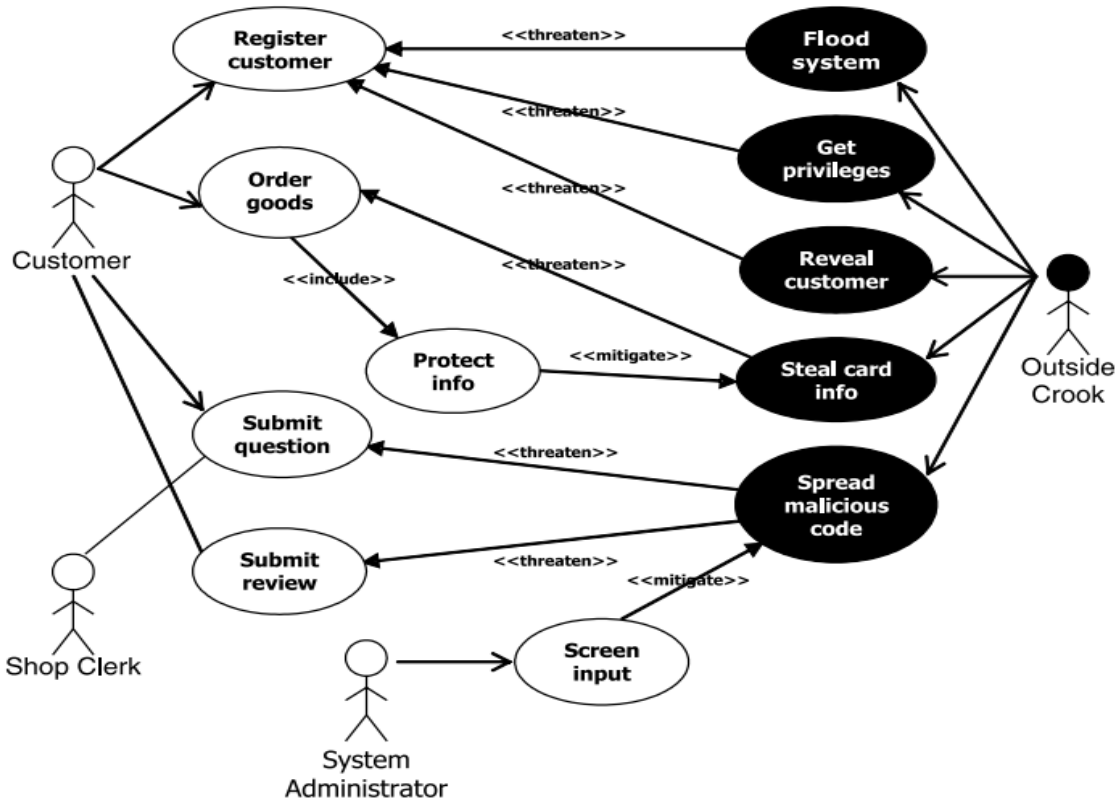


Fig. 1. Misuser (the black user) is threatening the use cases by initiating misuse cases (taken from [8])

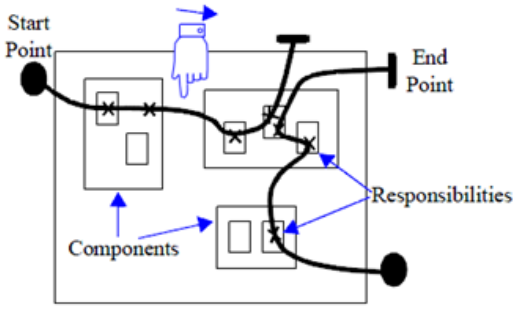
3.2.3. HARM: An approach toward secure architecture

As another view point in security, HARM (Hacker Attack Representation Method) makes a relation between the security requirements and the software architecture [11]. This approach looks at security requirements in the context of secure architectures. As the first step toward this goal, use case maps are used to show the use cases in the context of software architecture. They make a relation between the functional requirements and the architecture by mapping the use cases to the architectural components. In the next step, misuse case maps are introduced to show the misuse cases in the architectural context and tie up the security requirement to the architectural component. Misuse case maps play a major role in linking the security requirement and architecture. In the next section we are going to introduce UCMs and MUCMs which will lead us toward designing a secure architecture.

3.3. UCMs And MUCMs

Use case maps and misuse case maps are defined as a sort of complements to use cases and misuse cases. They try to show their precedence in an architectural context to clarify the binding of different architectural components to different use cases or misuse cases. These related modeling techniques in requirement engineering (use case maps and misuse case maps) used to model the behaviors and anti-behaviors of a software system and link them with the proposed software architecture.

Use case maps has been developed in Carlton University by Buhr and his team and used for describing and understanding of a wide range of systems since 1992 [2]. Use case maps were introduced to make a linkage between proposed system's behavior and its architectural structure, in a visual way. In most cases, there is a gap between basic requirements for the system and detail design. Use case maps are the best choice to fill this gap in requirement engineering and system design. Basic user activities are extracted from use cases which are then delegated to different architecture components. These delegations are modeled by a set of notations that can be classified in three main notation categories. Use case maps notations [1, 3, 4, 5] consist of scenario paths, architecture components and responsibilities.



Imagine tracing a path through a system of objects to explain a causal sequence, leaving behind a visual signature. Use Case Maps capture such sequences. They are composed of:

- **start points** (filled circles representing pre-conditions or triggering causes)
- causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- and **end points** (bars representing post-conditions or resulting effects).

The responsibilities can be bound to **components**, which are the entities or objects composing the system.

Fig. 2. Basic Notation and Interpretation for UCMs [3]

UCMs provide a combined overview of a software system's architecture and its behavior by drawing usage scenarios paths as lines across boxes that represent architectural run-time components. The boxes can be nested to indicate hierarchies of components. The scenario paths are connected to the components they run across by responsibilities drawn as crosses.

Fig. 2 shows a system which has three main architecture components (rectangles in the picture) and two scenario paths (thick black line with start and end points). Each scenario path uses architecture components to follow a specific scenario (is shown by drawing a scenario path through components).The scenario paths are connected to the components they run across by responsibilities (tasks, actions or functions to be performed by the component) drawn as crosses.

All available techniques and methods in security requirement engineering miss the relation with software architecture and look at the security issues from a point of view which has no idea of the architecture of the software. Misuse case maps (MUCMs) (Karpati et al., 2010) [2] present security issues from an architectural perspective. They combine perspectives from MUCs [8] and UCMs [1, 3, 4, 5]. MUCMs address security requirements by focusing on vulnerabilities, threats and intrusions (inherited from MUCs), from an architectural point of view (inherited from UCMs).

Fig. 3 shows the MUCM notations. These notations extend the UCM notations with some extra notation for intrusions. The intrusions are represented by one or more exploit paths. These exploit paths cut through vulnerable parts of the system. Each path starts with a triangle. If no damage happens, it ends in a bar or a "no entry" symbol. Otherwise, the exploit path ends in a lightning symbol. Exploit paths can be numbered as individual steps in a complex intrusion. The steps of an intrusion will usually be causally related, each one built on previous results. Responsibilities can be left out if they are not relevant for the overall intrusion. The system may have vulnerable points (such as authentication responsibility) or components (such as components without up-to-date security patches), which are suspect to threats. Mitigations can help to counter the threats and appear in the MUCM as desired possibilities. This translates to security requirements later. Misuses are depicted by the exploit path's crossing of a vulnerable point or part. Get, put ('+') and remove ('-') arrows can be used to show how an exploit path interacts with a component. An example of a put arrow is when the attacker ('arrow starting from the exploit path') installs ('+') a sniffer program on one of the servers ('arrow ends at the component'). (See [11] for more detail). In the following example, we will show how theses notations are used to model a real life intrusion scenario.

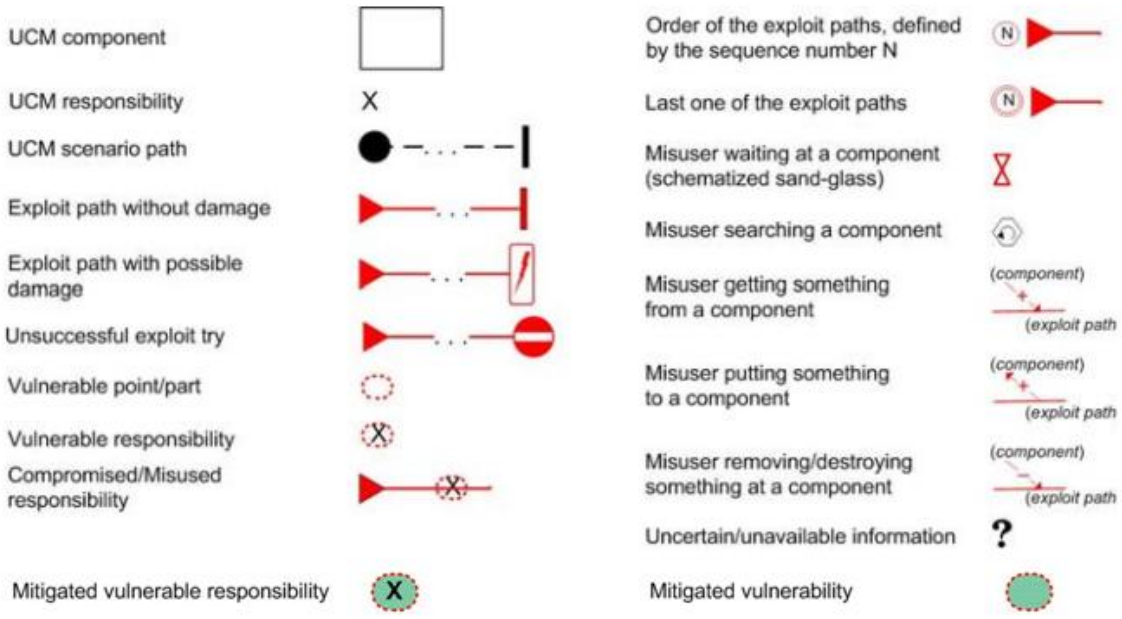


Fig. 3. MUCM notations [3]

Fig. 4 shows a real life intrusion scenario, it depicts the first 5 steps of a bank intrusion reported in the literature [19, Chapter 7] and modeled by MUCMs notations. First, the intruder found an interesting bank by browsing a web site with organizations and IP ranges assigned. Next, he probed for further details about the IP addresses of the bank and found a server that was running Citrix MetaFrame (remote access software). He then scanned other networked computers for the remote access port to Citrix terminal services (port 1494). The attacker knew he might be able to enter the server with no password, as the default Citrix setting is "no password required". He searched every file on the computer for the word "password" to find the clear text password for the bank's firewall. The attacker then tried to connect to routers and found one with default password. He added a firewall rule allowing incoming connections to port 1723 (VPN).

In This MUCM diagram (Fig. 4), the whole red line (the lines with numbers 1, 2, 3,4,5,7 are in red color) depicts the attacker's footprint whereas the dashed black line shows the regular users' activities. Arrows with plus sign show when the attacker puts/gets something from/to a component.

In the next section, we will take a look at related work (existent SRE editor tools), and will study different cases in order to find the best design and architecture to use in the development of our MUCM editor.

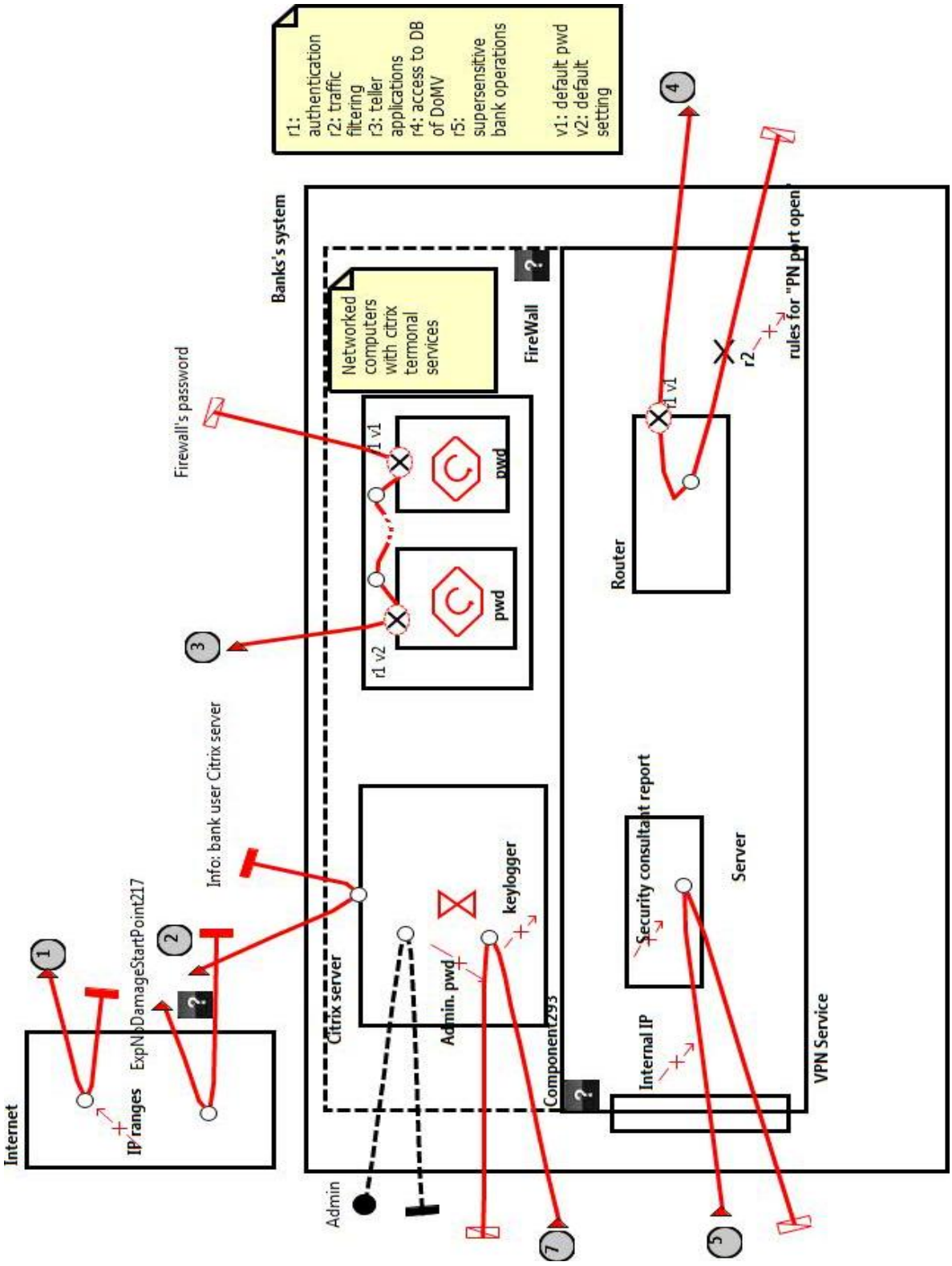


Fig. 4. A misuse case map created by the jUCMNav editor visualizes the first half of a bank intrusion.

4. Comparison of related tools

The following is an overview of the existing tools in SRE. The review of each tool starts with a little history followed by an investigation on the functionality and work flow. It will be concluded by a technical assessment on used technology, available technical documents and source code.

4.1. Surakasha security workbench

The first workbench in SRE, called Surakasha, has been developed by the National Institute of Technology Karnataka (NITK). The Surakasha security workbench [20] combines different methods in security requirement engineering and makes a similar mapping between the steps of functional requirement analysis and security requirement analysis as much as possible. As a case in point, functional requirement analysis starts with use cases and, in this security workbench's work flow, security requirement analysis starts with miss use cases. The work flow in Surakasha goes on with identifying important assets of the application to be developed. This identification should be done by brainstorming sessions between different stakeholders. After preparing a list of valuable assets for the application, each asset is viewed from three different perspectives as a customer, administrator and attacker. From each perspective every single asset gets a rank in three different major security principle areas called CIA (Confidentiality, Integrity, Availability) triads. As the second step, use cases will be developed to specify the functional requirements by a simple use case editor which is prepared by Surakasha. After that a lightweight misuse case diagram will be developed by another editor which is dedicated to this purpose in this workbench. Also another facility is prepared for making textual description of these misuse cases. As the next step, attack trees will be developed for each abstract threat in misuse cases. Surakasha uses DRAED analysis to measure the impact of each threat. The DREAD analysis rates the impact of threats by asking five fundamental questions about each threat: damage potential, reproductively, exploitability, affected users, and discoverability. Users should answer each question by assigning a numeric value between 0 and 10. Zero shows the lowest impact rate and 10 shows the highest impact for each question. DREAD risk value for each node of the attack tree will be calculated by the average rate of these five questions. In this way a DREAD impact rate will be calculated for each node starting from the leaf nodes and percolated up to the root node. Comparing the root node DREAD value with the cost of the asset will clear the mitigation plan and will classify the threats as 'considered' and 'neglected'.

Surakasha is an open source desktop application implemented in Java using the Swing GUI toolkit. Although it has a clean user interface, this software suffers from some serious deficiencies. It has a few bugs in saving and loading diagrams. Usability of the software is undesirable especially in working with attack trees. Since it has not sufficient technical documentation regarded the architecture, design and development extension and maintenance cost will be high for adopting this software. The quality of source code is quite poor with inappropriate packaging and insufficient comments and documentation.

In conclusion, we can see Surakasha as a workbench which offers support for asset identification and prioritization with some other facilities for making use case diagrams, misuse case diagrams, misuse case textual template, attack trees and DREAD analysis. It suffers from low software quality and insufficient technical documentation which impose high cost on the maintenance and extension.

4.2. SQUARE

The SQUARE methodology [13] was developed by cyber security lab in Carnegie Mellon University to support the nine-step Security Quality Requirement Engineering process. This process includes sub-processes and techniques to improve requirement identification, analysis, and specification. It also focuses on management issues associated with the development of good security requirements. The workflow consists of nine steps with a very rich and useful help document at the beginning of each step which describes purpose and functionality of that step. The workflow starts with identifying and defining common terms and definitions among the stakeholders to avoid any ambiguity between them during the project. The process continues with identifying prioritized assets and goals to be protected in the project. The next step is the collection of artifacts consisting of system architecture diagrams, use case diagrams, misuse case diagrams and attack trees. These artifacts are generated outside of the tool, and the tool will only hold a reference to them without preparing any facility for making these artifacts. The workflow continues with selecting elicitation techniques, eliciting security requirements, categorizing the requirements, prioritizing the requirements and finishes with inspecting the requirements. As a major difference with the Surakasha, the SQUARE tool does not prepare any facility and editor for making any special security related artifact such as use cases, misuse cases or attack trees. It should be considered as a managerial tool in security requirement engineering for increasing the quality during the process.

SQUARE is a web application developed in Java, with MySQL database and Tomcat Apache server. It is not an open source project, so the source code is not available. It is free to use on local machines or online [14] on the Carnegie Mellon website. End user help documents seem good but there is no technical document regarding the development.

In conclusion, SQUARE should be considered as a tool for software security requirement engineering process to reach the desired quality in this process. As a major difference compared to Surakasha, the SQUARE tool does not provide any facility and editor for making any special security related artifact such as use cases, misuse cases or attack trees. It should be considered as a managerial tool in security requirement engineering for increasing the quality of the process. Due to lack of source code, it is impossible to talk about the technical background and extending possibilities.

4.3. SeaMonster

SeaMonster [15, 16] is a security-modeling tool initiated by SINTEF and has been developed as a part of the SHIELDS project [17]. The SHIELDS project focuses on model-based detection and elimination of software vulnerabilities.

SeaMonster is a modeling tool for threat models and prepares two different editors for attack trees and misuse case diagrams. You can create and edit diagrams for both attack trees and misuse cases with related notations and perform related usual task such as loading and saving different diagrams.

SeaMonster is an open source desktop application which developed as an Eclipse plug-in and utilizes the Java programming language. Eclipse is an application platform which acts like a host for different plug-ins with divers functionality. Also it has been considered as one of the best open source IDEs for Java developers. SeaMonster utilizes three different modeling frameworks: Graphical Modeling Framework (GMF), Eclipse Modeling Framework (EMF) and Graphical Editing Framework (GEF). It has a simple and easy user interface with acceptable usability and reliability supported by good end user documentation. The extension and maintenance costs are not clear since there is no enough technical documentation related to development.

In conclusion, SeaMonster could be considered as a simple modeling tool for security requirements which supports attack trees and misuse case diagrams. It is developed based on the Eclipse platform with insufficient technical documentation which increases the cost of time and effort in extension.

4.4. jUCMNav

jUCMNav (the Java Use Case Map Navigator [18]) is part of user requirement notation (URN). URN is used for elicitation, analysis, specification, and validation of requirements. It consists of two complementary views, Goal-oriented requirement language (GRL) and use case maps (UCM). jUCMNav focuses on modeling for UCMs and supports all use case maps notations. It supports all fundamentals of the UCM such as paths, responsibilities and stubs. Also, there is possibility to activate and deactivate different scenario paths.

jUCMNav is an open source desktop application developed as an Eclipse plug-in and utilizes Java programming language. This is the most reliable and usable tool among all. It shares the Eclipse flexible user interface and professional features. It uses two different Eclipse development frame works and libraries, Eclipse Modeling Frame work (EMF) and Graphical Editing Framework (GEF). The administrator team of jUCMNav has prepared an on line SVN repository for the updated source code. The quality of the source code is accompanied by sufficient technical documentation related the architecture, used library and programming languages. On the other hand there are no low-level design documents such as class diagrams. It has different subproject under development such as UCM scenario project, Aspect-oriented User Requirements Notation project and Standard URN (Z.151) Import/Export project. These different extensions show its high potential in

extendibility. As the down side we can mention the complexity of the code and libraries which can slow down the development and impose additional time cost in development.

In conclusion, jUCMNav as a modeling tool for security requirements focuses on a single responsibility and prepares an editor for UCM. It developed based on Eclipse platform and takes the advantages of Eclipse professional user interface with a high software quality.

4.5. Remarks on the Comparison

We present in what follows a comparison of the previously mentioned tools with the aim of helping us to make a decision on which one to use as a base for our misuse case map editor.

SQUARE cannot be considered as a modeling tool, it is a managerial tool to reach high quality in SRE process. Surakasha, SeaMonster and jUCMNav are modeling tools and have good potential to be extended as a MUCM editor. jUCMNav which is working as a UCM editor is the most similar tool in functionality to a MUCM editor. It has the most common notations and functionalities with MUCM editor and seems a good choice to be extended. jUCMNav has been built based on the Eclipse platform as an Eclipse plug-in and took the advantages of the Eclipse professional user interface. It is very usable and reliable, sharing many features with MUCM editor but has the most code complexity between the other alternative tools. It seems that there is a tradeoff between complexity and usability. Table 1 shows the comparison of the preceding tools.

As seen in Table 1, Surakasha has the lowest usability and complexity, SeaMonster has a medium usability and complexity and jUCMNav has the highest usability and complexity. By considering the future extension of the MUCM editor as a HARM security workbench, we decided to use jUCMNav as the base platform for developing the MUCM editor.

Table. 1. Comparison of the tools in SRE

Application	Open source	Type of application	Programming language	Used libraries	Technical documents	Usability	Reliability	Portability	Code quality	Complexity of code
Surakasha	Yes	Desktop	Java	Swing	Low	Low	Low	High	Low	Low
SQUARE	No	web	---	---	Low	Medium	High	High	---	---
SeaMonster	yes	Eclipse base	Java	EMF, GMF, GEF	Low	Medium	High	High	Medium	Medium
jUCMNav	yes	Eclipse plugin	Java	EMF, GEF	High	High	High	High	High	High

5. MUCM editor

We had two options for developing a MUCM editor: the first option was developing from scratch and the second option was extending an existent tool. Based on our results, shown on Table 1, we decided to extend jUCMNav as an editor for MUCMs. We name it jMUCMNav.

In jMUCMNav is possible to draw an intrusion scenario using MUCM notations as drawing components. These notations are placed on the right hand side palette of the editor

(see Fig. 5) appearing as an Eclipse palette after creating a new misuse case map diagram through the file menu. We categorized the notation in different drawers inside the palette based on their functionality.

One can drag and drop the elements of the notation from the palettes to the main editor screen, bind them together and move them on the screen. Every path consists of an end point, start point, linking edges and empty points. It is possible to change the shape of the path by dragging the empty point on the screen. Furthermore you can attach other elements to the path in these empty points, but the editor does not perform any syntactic or semantic check on how the elements are put together.

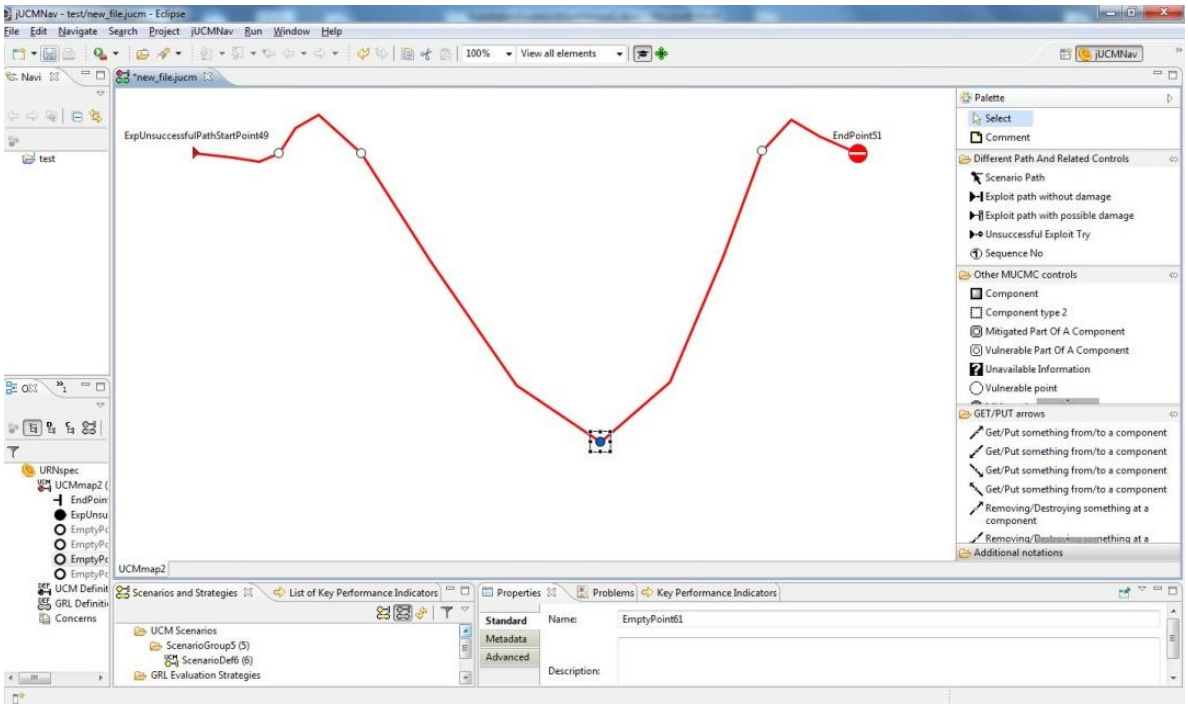


Fig. 5. jMUCM editor

5.1. Architecture

The jMUCMNav editor [12] has been developed as an Eclipse plug-in and needs Eclipse as its host application. It is developed in Java and uses a model driven architecture. EMF (Eclipse Modeling Framework) and GEF (Graphical Modeling Framework) are used in the development of the jMUCMNav to build a model driven architecture. By defining a base model, the EMF is able to generate the basic code for our MUCM editor. A model in jMUCMNav is defined as the notations for misuse case maps [2], parent and child relation among them and the hierarchy of the notations. The code generation facility of the EMF on a defined model provides high modifiability and extendibility for the MUCM editor.

5.2. User interface and Usability

The jMUCMNav interface inherited from the professional user interface of Eclipse. Eclipse is one of the most user-friendly IDEs (Integrated Development Environments) which have popular characteristics in user interface design with UI pallets, views and perspectives. Since jUCMNav gets its UI features from Eclipse, our discussion about the jMUCMNav user interface will end with a criticism of the Eclipse user interface. As the first evaluation of the usability we can rely on the usability of Eclipse and jUCMNav which are the base of jMUCMNav. Since we have changed only the notation of jUCMNav, we can claim that we inherited all the usability features of the jUCMNav and Eclipse IDE.

Next section shows the result of an experiment to evaluate efficiency and usability of the editor. In this experiment, we compare our newly developed editor with other alternative editor tools.

6. Validation of jMUCMNav

In order to validate jMUCMNav, we focus on its usability and efficiency. Validation of jMUCMNav is accomplished by analyzing the data which is gathered by questionnaires during an experiment. In this experiment performance is measured as scale for efficiency and ease of use is measured as scale for usability. During this experiment we asked people to fill in a questionnaire after modeling two real world intrusion scenarios. We asked them to use jMUCMNav and another alternative editor for modeling, and make a comparison between these tools.

6.1. Experiments design

To validate the usability and efficiency of the editor, we conducted an experiment with eight people who used the MUCM editor with another general purpose editor to (e.g. Microsoft paint) individually model two different real world intrusion cases (Bank hack [21] and Penetration test [21], fully descripted in Appendix D and E) with MUCM notations.

To control the order in which the techniques were used and the cases were solved, a Latin-Squares experimental design was used as shown in Table 2. In the table, we have two different intrusion cases, two different editors and four groups of people. Each row of the Table 2 shows the order in which the editors should be used, and each column shows the order in which the cases should be solved. For example, in the first step, Group A should use jMUCMNav for modeling the bank hack intrusion case, and then it should use an alternative editor to model the penetration test intrusion case.

Table. 2. Latin-Squares experimental design used in the experiment

Case order: Technique order:	Bank hack before penetration test	Penetration test before bank hack
jMUCMNav before alternative editor	Group A	Group B
Alternative editor before jMUCMNav	Group C	Group D

During this experiment, after controlling the participants' background, three types of tasks were solved: an understanding-task, a performance-task and a perception-task. The background was measured by a pre-task questionnaire addressing the participants' self-assessed knowledge of requirement analysis, modeling, coding, testing, etc. They were also asked to report the number of completed semesters of ICT studies and months of ICT-relevant work experience. The pre-task questionnaires are represented in the Appendix B.

For the "understanding part" of the experiment, we placed two sections as reading tasks for the participants to help them understand the MUCM modeling language and jMUCMNav. The first part is a brief introduction to MUCM modeling language and its notations and the second part is the jMUCMNav user manual. You can find these parts of the experiment sheets in appendix A and D.

Performance was measured by asking the participants to identify and list the vulnerabilities and mitigations of two different intrusion cases by using misuse case map editor and an alternative editor. The number of identified vulnerabilities and mitigation was counted as the main factor for efficiency and usefulness of the two editors.

Perception is measured by a post-task questionnaire. Technology Acceptance Model (TAM) [22, 23] was used for designing the post-task questionnaire in a way that four questions addressed perceived usefulness (PU), four addressed perceived ease of use (PEOU) and four investigated the participants' intention to use (ITU) the technique in the future.

In the following section we introduce a list of the different variables used in the analysis of the experiment.

6.2. Variables of the experiment

Table 3 summarizes the main variables used in the analysis of the experiment. The first column shows the name of the variable and the second column shows a short description of the variable name. For example, to assess different background knowledge of the participants, we use variables which start with KNOW followed by an underline and two letter abbreviation of a specific knowledge. For instance, we use KNOW_ST to represent the background knowledge of the participant in Software Testing.

6.3. Hypotheses

The hypotheses of the experiment are listed in Table 4. The first column of the table shows the name of the hypothesis, the second column shows a little description and the last column is an abbreviation. For example, we named the first hypothesis as H1. It shows that if we model an intrusion case with two different editors (jMUCMNav and the other alternative editor), we will identify different numbers of vulnerabilities. It shows different performance between the editors.

Table 3. variables used in the experiments

Name	Explanation
EDITOR=MUCM EDITOR =ALTE	The editor used in that part of the experiment, either MUCM editor or desired alternative editor.
CASE=BANK, CASE=PEN	The case solved in that part of the experiment, either BANK (the bank intrusion) or PEN (the penetration test)
KNOW_RA, KNOW_SD, KNOW_SC, KNOW_ST, KNOW_SM, KNOW_SEC KNOW_UC, KNOW_MUC, KNOW_UCM, KNOW_MUCM, KNOW_ALTE,	The participants' self-assessed knowledge about requirement analysis (KNOW_RA), software designing(KNOW_SD), coding (KNOW_SC), testing(KNOW_ST), modeling(KNOW_SM) , security analysis(KNOW_SEC) , use cases(KNOW_UC) , misuse cases (KNOW_MUC) , use case maps(KNOW_UCM), Misuse case maps(KNOW_MUCM) and their desired alternative editor(KNOW_ALTE) on a 5-point scale, where 1 is “Never heard about it” and 5 is “Expert”.
STUDY	The participants' self-reported semesters of ICT-studies.
JOB	The participants' self-reported months of ICT-relevant work experience.
VULN MITIG	The numbers of unique vulnerabilities and mitigations identified by the participants.
VUMI	The sum of unique vulnerabilities and mitigations identified by the participants.
PER_PU, PER_PEOU, PER_ITU	Average scores on the 5-point Likert scales for the four statements about perceived usefulness of, perceived ease of use of and intention to use of the techniques.
PER_AVE	Average scores on the 5-point Likert scales for all the twelve statements about the techniques.

6.4. Experiment procedure

The eight participants of the experiment were selected from people with different background in information technology and computer science. These eight participants were divided into four groups each of which consisted of two people solving the tasks described in the experiment sheet and distributed by email to the individual participants. The experiment comprised 10 steps:

1. Reading the "Introduction and guidelines to the experiment"
2. Introduction to the MUCM notation
3. Reading the "Installation of tool(s) and user manual" and installing the tool(s)
4. Pre-experiment questionnaire about background
5. Using one of the programs with one of the intrusion cases
6. Eliciting vulnerabilities and mitigations
7. Estimating diagram use
8. Using the other program with the other intrusion case
9. Eliciting vulnerabilities and mitigations
10. Estimating diagram use
11. Comparison questionnaire
12. Opinions and comments

In the following section we will show the result of the experiment in four different parts. We will start by comparing the background of the participants and will continue with comparing two editors based on three main areas of the TAM: perceived usefulness (PU), perceived ease of use (PEOU), intention to use (ITU) the technique in the future.

Table 4. Hypotheses of the comparison experiment

H1	There will be a significant difference in the number of identified vulnerabilities between the diagrams produced by jMUCMNav and alternative Editor.	VULN[MUCM] ≠ VULN[ALTE]
H2	There will be a significant difference in the number of identified mitigations between the diagrams produced by jMUCMNav and alternative Editor.	MITI[MUCM] ≠ MITI[ALTE]
H3	There will be a significant difference in the number of identified vulnerabilities and mitigations between the diagrams produced by jMUCMNav and alternative Editor.	VUMI[MUCM] ≠ VUMI[ALTE]

H4	The usefulness of jMUCMNav and alternative editor will be perceived differently.	PER_PU[MUCM] ≠ PER_PU[ALTE]
H5	The ease of use of MUCMs and alternative editor will be perceived differently.	PER_PEOU[MUCM] ≠ PER_PEOU[ALTE]
H6	The intentions to use toward MUCMs and alternative editor will be different	PER_ITU[MUCM] ≠ PER_ITU[ALTE]
H7	MUCMs and alternative editor will be perceived differently.	PER_AVE[MUCM] ≠ PER_AVE[ALTE]

6.5. Experiment Results

In the beginning, the experiment was planned to start with twenty participants in four different groups with a series of systematic statistical data analysis methods like Kruskal-Wallis H test [24] and Wilcoxon signed-rank tests [25]. Due to the complexity of the experiment, we couldn't reach enough volunteer participants. The small size of the groups made it impossible to use systematic statistical data analysis method like Kruskal-Wallis test which need a minimum of five people in each group. Therefore, we decided to simply compare the data in different groups and use the mean and standard deviation to reach a rough conclusion about the efficiency of the software.

6.5.1. Comparing background

To compare differences between the four groups of participants, different background variables were used as shown in Table 5. These variables rate group's expertise in different areas where 1 is "Never heard of it" and 5 is "Expert". Every column shows an expertise and every row shows one of the groups. Each of the cells shows the average of a special expertise in a special group.

Table 5. Background comparison

Background variable Group	**KNOW_RA**	**KNOW_SD**	**KNOW_SC**	**KNOW_ST**	**KNOW_SM**	**KNOW_SEC**
A	3.5	3	3.5	3	3	2.5
B	3	3	4	3	4	2
C	4	4	4	4	4	4
D	3.5	4.5	5	3.5	3.5	2.5

Table 5. Background comparison (continue)

Background variable Group	**KNOW_UC**	**KNOW_MUC**	**KNOW_UCM**	**KNOW_MUCM**	**KNOW_ALTE**
A	4	2	2	2	2.5
B	3	1	3	1	4
C	3	3	2	1	3
D	3.5	1.5	1.5	1.5	4

Participants reported being significantly more knowledgeable about requirement analysis, software coding, modeling and testing. Minimum knowledge average in groups belongs to misuse cases and misuse case maps. The overall comparison shows that there is no significant difference between groups, all of them have a maximum average in software modeling and minimum average of knowledge in use case maps and misuse case maps.

The participants reported between one and 20 years of IT related work experience with the average of 6.7 years. It shows that both junior IT students as well as experienced professors are placed in experiment to reach more accurate results.

6.5.2. Performance

The performance of the editors depends on their abilities to reveal vulnerabilities and mitigations. The numbers of identified vulnerabilities and mitigations are counted in the different diagrams produced by jMUCMNav and the other alternative editor, and then the mean and standard deviation are calculated to compare how well the participants identified vulnerabilities and mitigations by using these two editors. Table 6 shows the mean and the standard deviation of the identified vulnerabilities and mitigations for jMUCMNav and the other alternative editor.

The results show no significant differences in the number of identified vulnerabilities and mitigations by means of two different editors, thus rejecting hypothesis H1, H2 and H3. Rejecting these hypotheses shows that when we finished modeling MUCM diagrams by any desired editor, there will be no differences in the numbers of identified vulnerabilities and mitigations based on the drawn diagrams.

We did not attempt to rate, categorize or analyze the vulnerabilities and mitigations in further detail, leaving this for further work. In particular, further work should investigate whether there are systematic differences between the types of vulnerabilities and mitigations identified using the diagrams drawn by the two different editors.

Table 6. Comparison results for performance

Identification task	Performance			
	jUCMNav Editor Mean	jUCMNav Editor St.Dev	Alternative Editor Mean	Alternative Editor St.Dev
Vulnerabilities (VULN)	3.88	0.74	3.88	1.24
Mitigations (MITI)	3.88	0.74	3.88	1.24
Both (VUMI)	3.88	0.74	3.88	1.24

6.5.3. Perception

Finally, a post task questionnaire (Appendix C) is designed to perceive the future of the jMUCMNav editor. It uses the technology acceptance model (TAM) to compare how the participants perceived the editor in terms of usefulness and ease of use, and whether they intended to use the editor again in the future. Each Aspect of the TAM, are represented by a TAM variable and evaluated by two questions, each rated on a 5-point Likert scale.

The calculated mean and standard deviation (St.Dev) for all of TAM variables are shown in Table 7 for both editors. Each row of the table shows a TAM variable and each column shows the mean or standard deviation for a specific editor. The participants perceived jMUCMNav significantly more positive than their alternative editor, for perceived usefulness, perceived ease of use and intention to use. Perceived ease of use is the best result among the other TAM variables which will affect the intention of use in the future. In conclusion, the average of all perception variables for jMUCMNav editor shows better result in comparison with any other alternative editor. Hence, hypothesis H4, H5, H6 and H7 are all confirmed.

Table 7. Comparison results for perception

TAM Variable	Perception			
	jUCMNav Editor Mean	jUCMNav Editor St.Dev	Alternative Editor Mean	Alternative Editor St.Dev
Perceived usefulness (PU)	3.56	0.57	1.94	0.07
Perceived ease of use (PEOU)	4.25	0.82	2.18	1.17
Intention to use (ITU)	4	0.46	2.13	0.64
Average	3.94	0.62	2.08	0.63

6.6. Remarks on the experiment

The experimental comparison indicates that the jMUCMNav editor and the alternative editor are equal in performance. They encourage users to identify equal numbers of vulnerabilities and mitigations in their target systems. Further analysis is needed to investigate if they encourage identifying the same types of vulnerabilities and mitigations.

However, jUCMNav is perceived more positively by users, i.e., they rate jMUCMNav more highly in terms of perceived usefulness, perceived ease of use and intention to use. A summary of the decisions about the hypotheses can be seen in Table 8. It shows the accepted and rejected hypotheses based the result of the experiment.

Table 8. Results of hypothesis testing

H1	H2	H3	H4	H5	H6	H7
Rejected	Rejected	Rejected	Accepted	Accepted	Accepted	Accepted

Since the jUCMNav editor is the first editor developed for making misuse case maps, it was not possible to compare it with other similar editors. Therefore, we had to make the comparison with a general purpose editor based on the user's preferences such as Microsoft paint. Comparison with a user desired editor means that we are competing with an editor that the user is more comfortable with, and this could affect the results.

7. Conclusion

This thesis work presented an editor for the misuse case maps security-modeling technique. The technique was introduced, and the choices regarding the editor implementation elaborated. Finally, the usability, appropriateness and efficiency of jMUCMNav are evaluated after implementation.

A new method in SRE has been introduced which looks at the security issues from a different point of view. It looks at the security issues in the context of architecture by introducing and making the benefits of misuse case maps which makes a relation between misuse cases and architectural component. This new approach uses a few methods and techniques like use case maps, attack tree diagrams and attack sequence diagrams, plus misuse case maps to introduce the HARM (hacker attack representation method). We need a workbench to support this approach and help security engineers in every single step by preparing the appropriate facilities and editors. Originally, jMUCMNav was developed as a misuse case map editor for HARM workbench, based on jUCMNav which is an Eclipse based use case map editor. By using the Eclipse Modeling Framework and Eclipse as an infrastructure, we took the advantage of model driven architecture and reached quite reliable software with an acceptable degree of **extendibility**.

The **usability and efficiency** of the jMUCMNav editor were measured by an experiment which used Latin square experiment design and technology acceptance model. The main measured factors during the experiment were performance, usefulness, ease of use and

intention to use. By comparing these factors between editors, we found out that the performance and number of security vulnerabilities and mitigations which we detected in the target system was not related to the editors we used to model. Furthermore, the experiment results showed that the usability and ease of use of the jMUCMNav editor is more positive than any other editor in MUCM modeling.

As a future work plan, the jMUCMNav editor will be integrated into a bigger workbench which will be produced for supporting the Hacker Attack Representation Method (HARM). HARM aims to be a security requirements engineering methodology using the attacks as starting point, and this workbench will provide some facilities to prepare UCMs, attack sequence diagrams, attack trees and attack patterns [11].

Developing the first editor which supports misuse case maps modeling, is one big step forward in security modeling to help the growing need for security requirement analysis and security enabled software.

References

1. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Transactions On Software Engineering 24(12), 1131–1155 (1998)
2. Karpati P., Sindre G., Opdahl A. L. (2010), Visualizing Cyber Attacks with Misuse Case Maps, In: LNCS Lecture notes in artificial intelligence; Volum 6182. pp. 262-275
3. Amyot, D., Use Case Maps Quick Tutorial. See http://www. usecasemaps. org/pub/UCMtutorial/UCMtutorial. pdf, 1999.
4. Buhr, R., Casselman, and R.: Use case maps for object-oriented systems. Prentice-Hall, Inc., Upper Saddle River (1995)
5. Buhr, R.J.A.: Use case maps for attributing behavior to system architecture. In: 4th International Workshop of Parallel and Distributed Real-Time Systems (1996)
6. Rannenberg K, Pfitzmann A, Müller G (1999) IT security and multilateral security. In: Müller G, Rannenberg K (eds) Multilateral security in communications—technology, infrastructure.Economy Addison-Wesley, pp 21–29
7. B.Fabian, S. F. Gürses, M. Heisel, T. Santen, H. Schmidt: A comparison of security requirements engineering methods. Requir. Eng. 15(1): 7-40 (2010)
8. Sindre, G. and A.L. Opdahl, Eliciting security requirements with misuse cases. Requirements Engineering, 2005. 10(1): p. 34-44.
9. Lodderstedt T, Basin DA, Doser J (2002) SecureUML: a UMLbased modeling language for model-driven security. In: Proceedings of the 5th international conference on the unified modeling language (UML'02). Springer, London, pp 426–441
10. Jürjens J (2003) Secure systems development with UML.Springer, New York
11. Karpati P., Sindre G., Opdahl A. L (2010), Towards a Hacker Attack Representation Method. Proc. of the 5th ICSOFT, INSTICC Press, pp. 92-101
12. http://code.google.com/p/mucm/
13. Nancy R. Mead and Ted Stehney. 2005. Security quality requirements engineering (SQUARE) methodology. SIGSOFT Softw. Eng. Notes 30, 4 (May 2005), 1-7.
14. https://squaretool.cylab.cmu.edu/Square/
15. Tøndel, I.A., Jensen, J., Røstad, L. (2010) Combining misuse cases with attack trees and security activity models, Proc. ARES 2010, pp.438-445.
16. http://sourceforge.net/apps/mediawiki/seamonster/
17. http://www.shields-project.eu/
18. http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome
19. Mitnick, K.D. and W.L. Simon, The art of intrusion: the real stories behind the exploits of hackers, intruders & deceivers. 2005: Wiley.
20. Maurya, S., Jangam, E., Talukder, M., Pais, A.R. (2009) Suraksha: A security designers' workbench. Proc. Hack.in 2009, pp. 59–66.
21. Mitnick, K.D., Simon W. L.: The Art of Intrusion. Wiley Publishing Inc (2006)
22. Davis, F.D.: Perceived usefulness, perceived ease of use and user acceptance of information technology. MIS Quarterly 13, pp. 319–340 (1989)
23. Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D.: User acceptance of information technology: Toward a unified view. MIS Quarterly, 27(3), pp. 425-478 (2003)
24. http://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_one-way_analysis_of_variance
25. http://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

Appendix A: User manual

1. Introduction

This document will walk you through steps required to install and use jMUCMNav, a misuse case map navigator. It is a graphical editor for creating misuse case map diagrams. It has been developed as a part of a master thesis called "Visualizing Cyber Attacks with Misuse Case Maps" by Yashar Bizhanzadeh[1].

1.1. Useful links and resources

- jUCMNav- home page , http://lotos.csi.uottawa.ca/ucm/bin/view/ProjetSEG/WebHome
- jMUCMNav - home page, http://code.google.com/p/mucm/
- Eclipse -home page, http://eclipse.org/

1.2. Prerequisites

JMUCMNav application has been developed as an Eclipse plug-in and needs Eclipse as its host application. In the first step, you have to install Eclipse. You can find different versions of Eclipse for different platform (Windows, Linux, Mac) at this URL http://www.eclipse.org/downloads/. For example, if you are using an ordinary 32 bit Windows operating system, try to download "**Eclipse IDE for Java EE Developers"** by choosing windows operating system from the upper right corner combo box and select 32 bit version for download. (See Fig. 1)

Next, you need Java SE 5 or greater to run Eclipse and its plug-ins on your system. Eclipse uses its own built-in Java complier but it is also possible to use any other external Java compiler. The type of operating system affects the type of Java and Eclipse which you want to install. For example, 32bit and 64bit operating systems have their own versions of Java compilers and Eclipse. After installing appropriate versions of Eclipse and Java, you will be able to install any plug-ins on your Eclipse.

32 bit windows operating system, Eclipse Galileo and Java SE 6 is used for all development, testing and installation of this project.

[1] In cooperation with Chalmers, Gothenburg and NTNU Universities under supervision of Gerardo Schneider, Guttorm Sindre and Peter Karpati.

Fig. 1. Eclipse download link

After downloading the appropriate version of Eclipse, you need to unzip the application. It doesn't need any special installation. If you are faced with any problem in installation process please refer to Eclipse official web site at http://www.eclipse.org/.

2. Installation Guide

After installing Eclipse, you are going to install JMUCMNav as a plug-in for it.

2.1. First time installation

For a successful installation please go through the following steps.

- Run Eclipse application.
- Select "install new software" form the help menu. (Fig. 2)
- Installation window should appear. (Fig. 3)
- Select "Add" button on installation window. (Fig. 3)
- Add URL "https://mucm.googlecode.com/svn/trunk/seg.jUCMNav.updatesite" at location text box and enter your desired name. (Fig. 4)

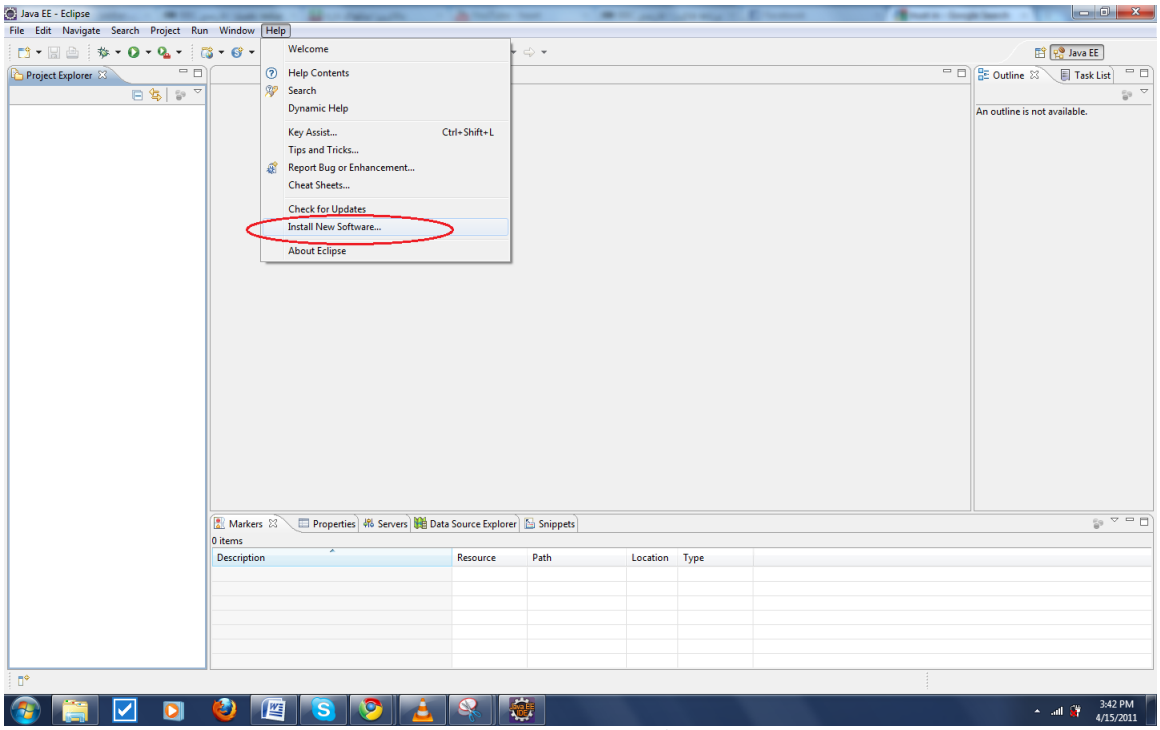


Fig. 2. Install new software

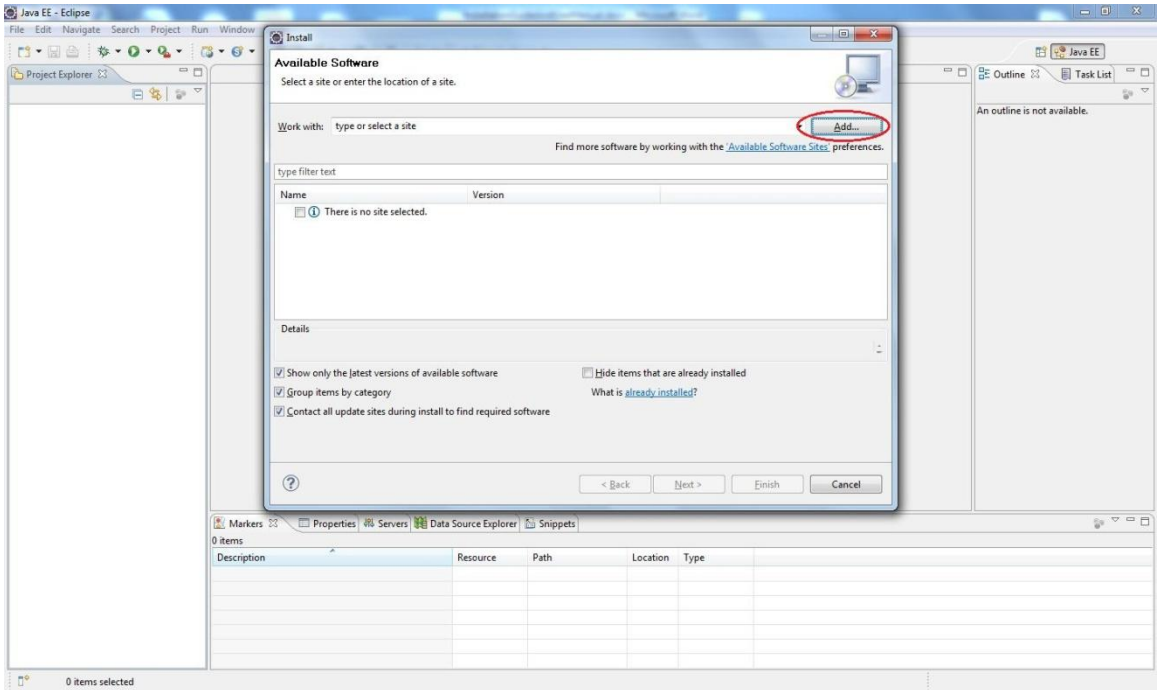


Fig. 3. "Add" button on installation window

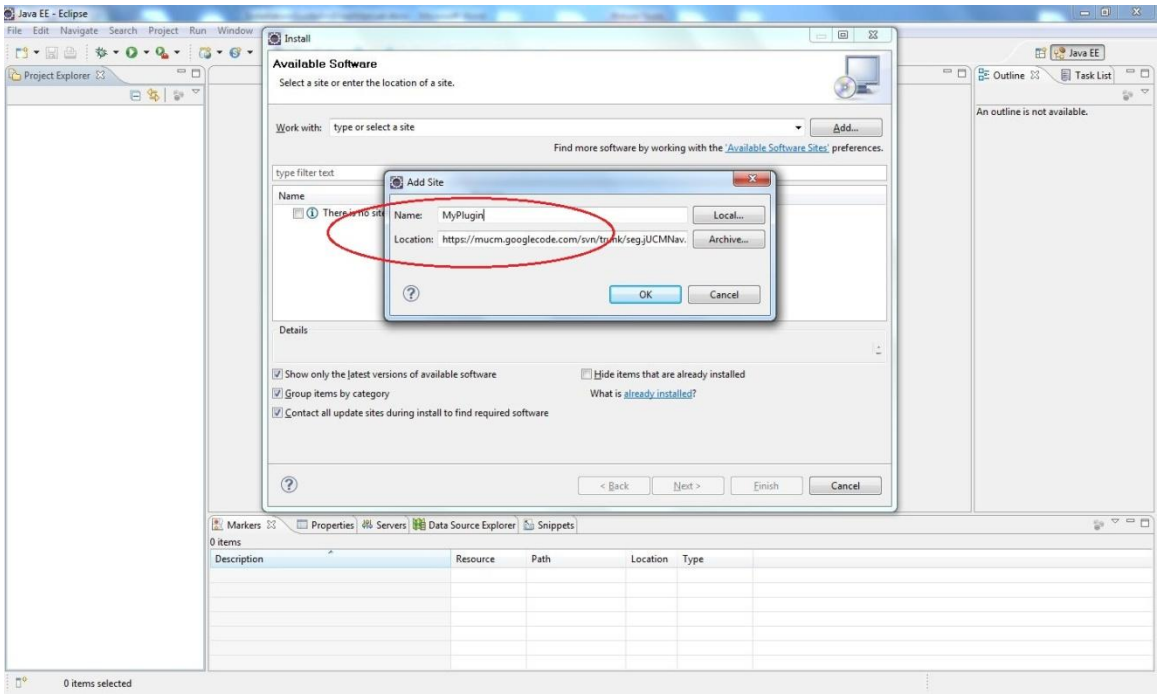


Fig. 4. Installation URL

- Click on OK button and close the dialog box.
- After a few second, the plug-in name will be displayed on installation window. (jMUCMNav) (Fig. 5)
- Select name of the plug-in and click on next button. (Fig. 5)
- The check box on the left bottom corner, called “Contact all update sites during install to find required software”, should be checked for installing required third party libraries. (Fig. 5)
- If there are any additional steps, conduct the process forward by simply clicking on the next button.
- Accept the terms of the license agreements and click on Finish button. (Fig. 6)

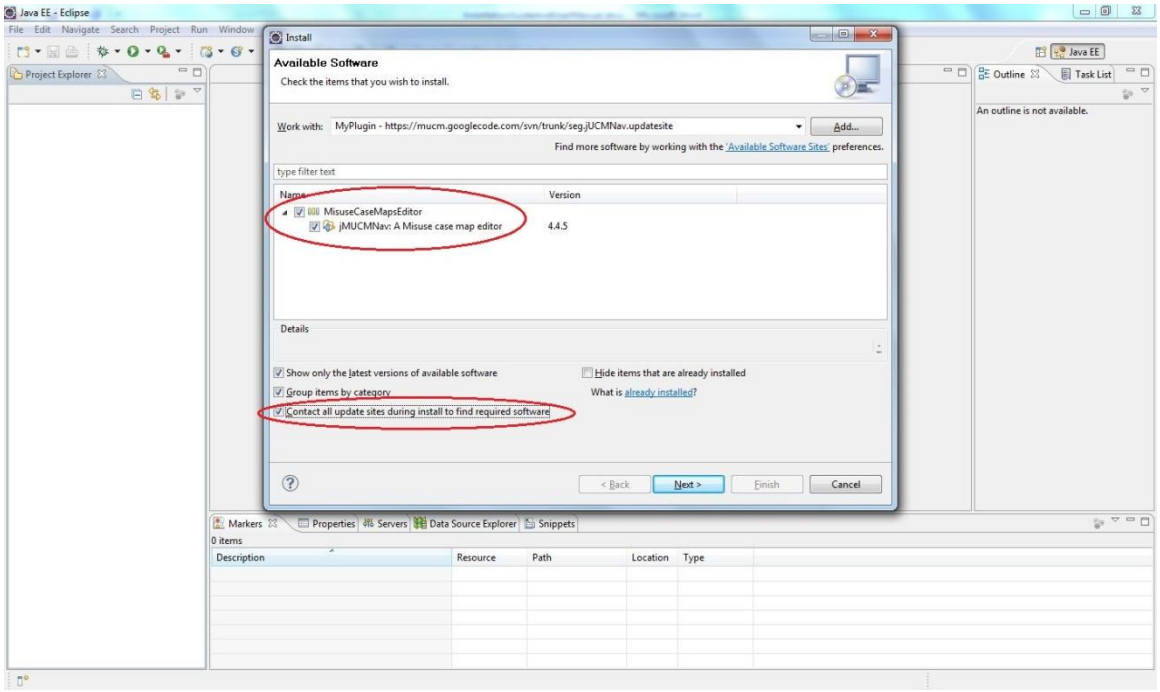


Fig. 5. Installation window

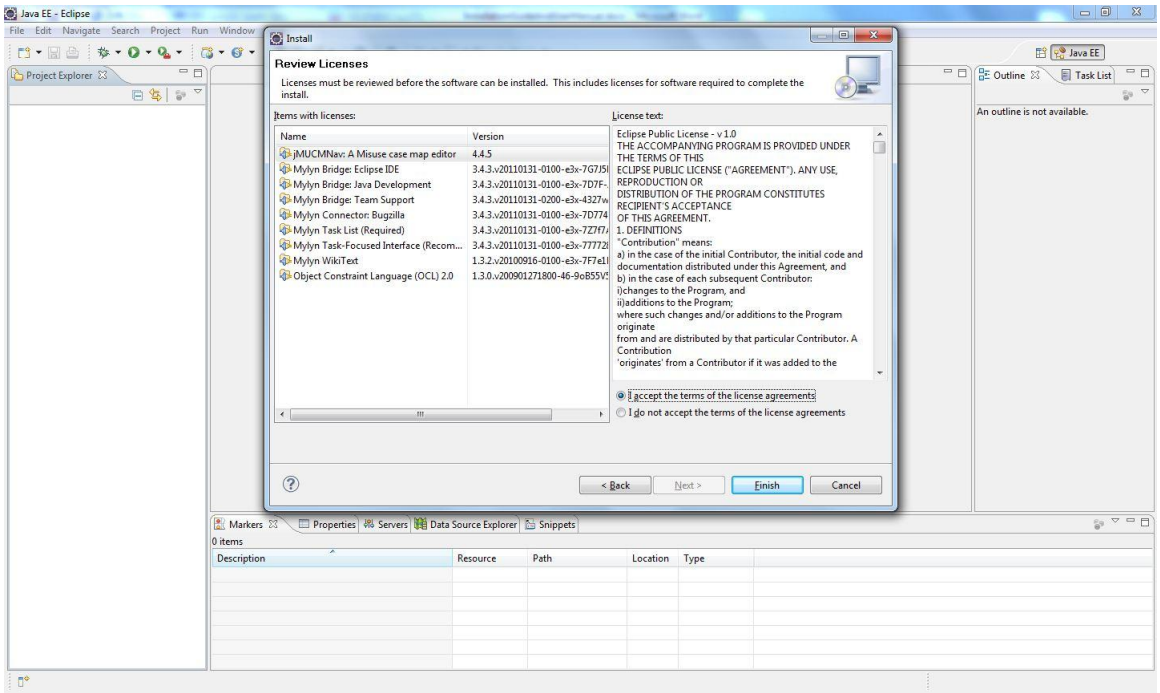


Fig. 6. License agreements

- Installation will take a few minutes based on the internet connection and SVN server speed.
- You have to trust in some certificates and security warning to complete the installation. (Fig. 7)

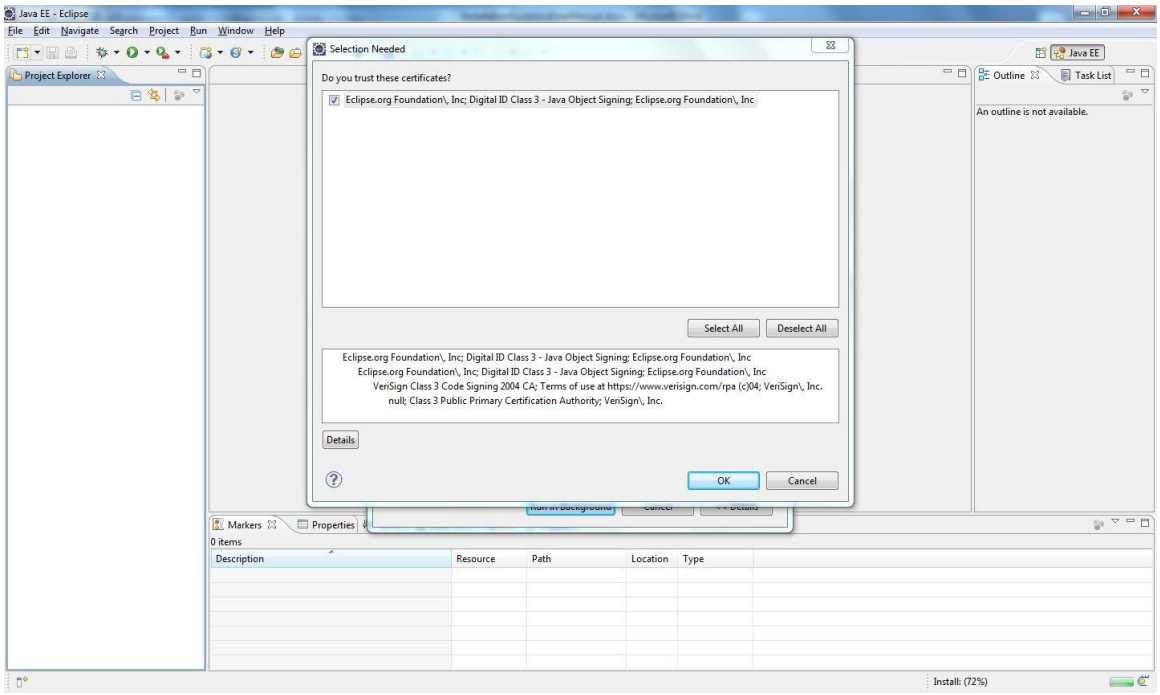


Fig. 7. Security warning

- Finally Restart your Eclipse.

2.2. Upgrading to a newer version

For upgrading the plug-in to a newer version, please go through the following steps

- Run Eclipse application.
- Select “Install new software” from help menu.
- Click on the link called “What is already installed?” (Fig. 8)
- “Eclipse installation details” window will appear. (Fig. 9)
- Select the plug-in name (jMUCMNav) from “installed software” tab and click on update button. (Fig. 9)
- Confirm the available update dialog box and click on the Finish button to start upgrading.
- Finally restart your Eclipse.

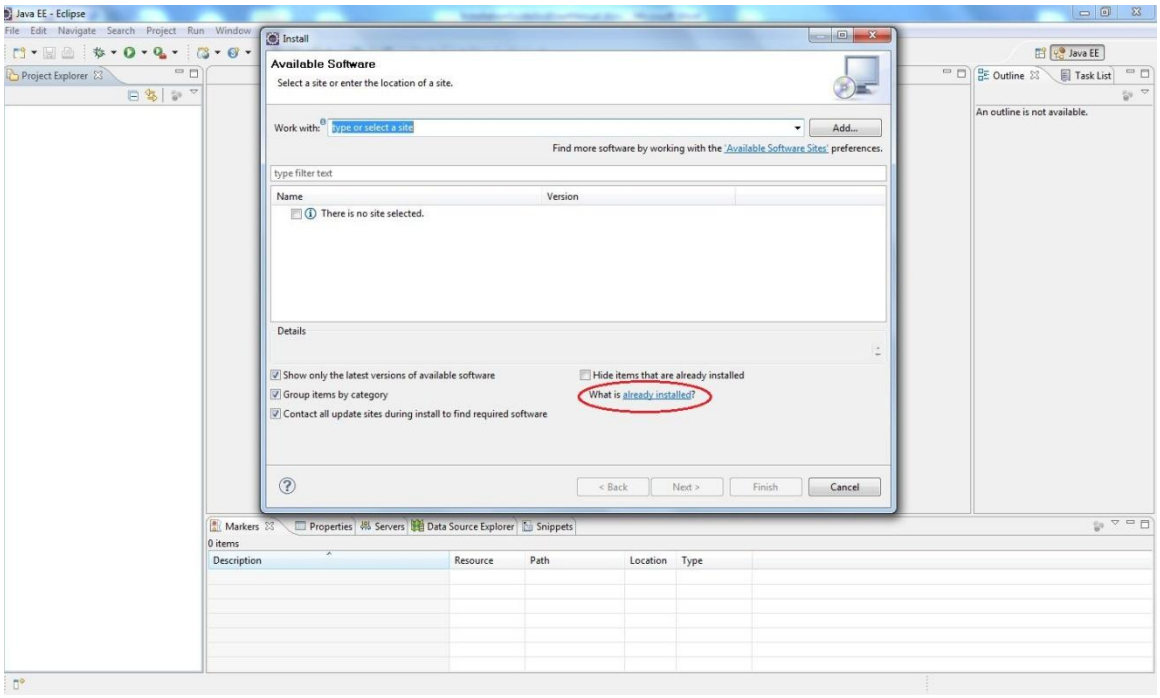


Fig. 8. Link to installed applications

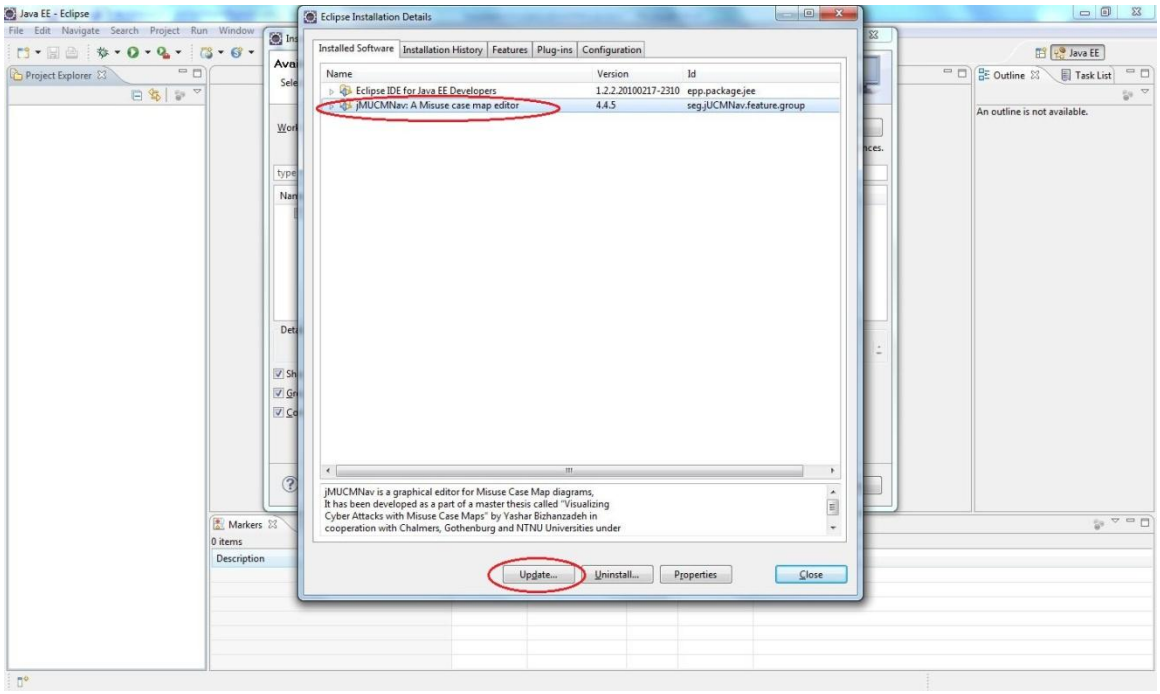


Fig. 9. "Installation details" window

3. Usage

In this section we are going to walk through the main use-cases of JMUCMNav.

3.1. Creating new Diagram

MUCM diagrams should belong to an existing project or folder in your workspace. For creating a new misuse case map diagram, please go through the following steps.

- Create a new project or folder, if you don't have any existing folder or project in your workspace.
- From "file" menu select the "new" then "Other..." submenus. (Fig. 10)

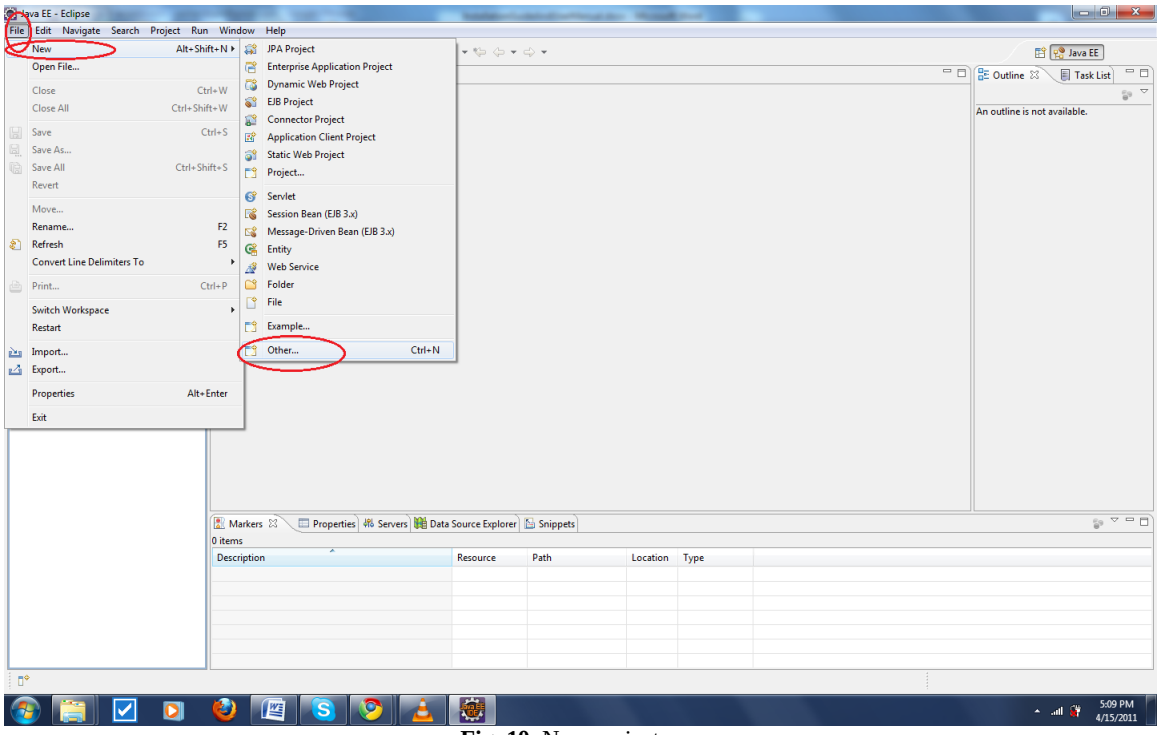


Fig. 10. New project menu

- A wizard will appear.
- Select jUCMNav as a resource in the wizard and click the "next" button. (Fig. 11)
- Select an existing folder or project in your work space, and then give a name to your misuse case map diagram. (Fig. 12)
- If you don't have any existing folder or project in your work space, go back and create it first.

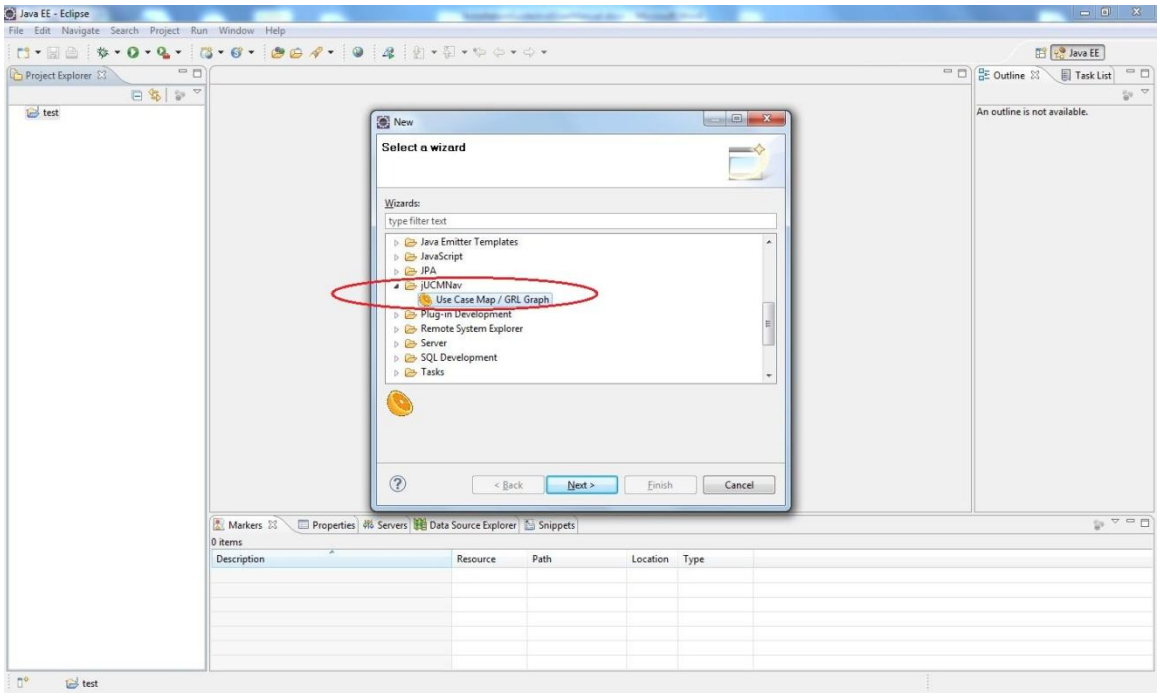


Fig. 11. New project wizard

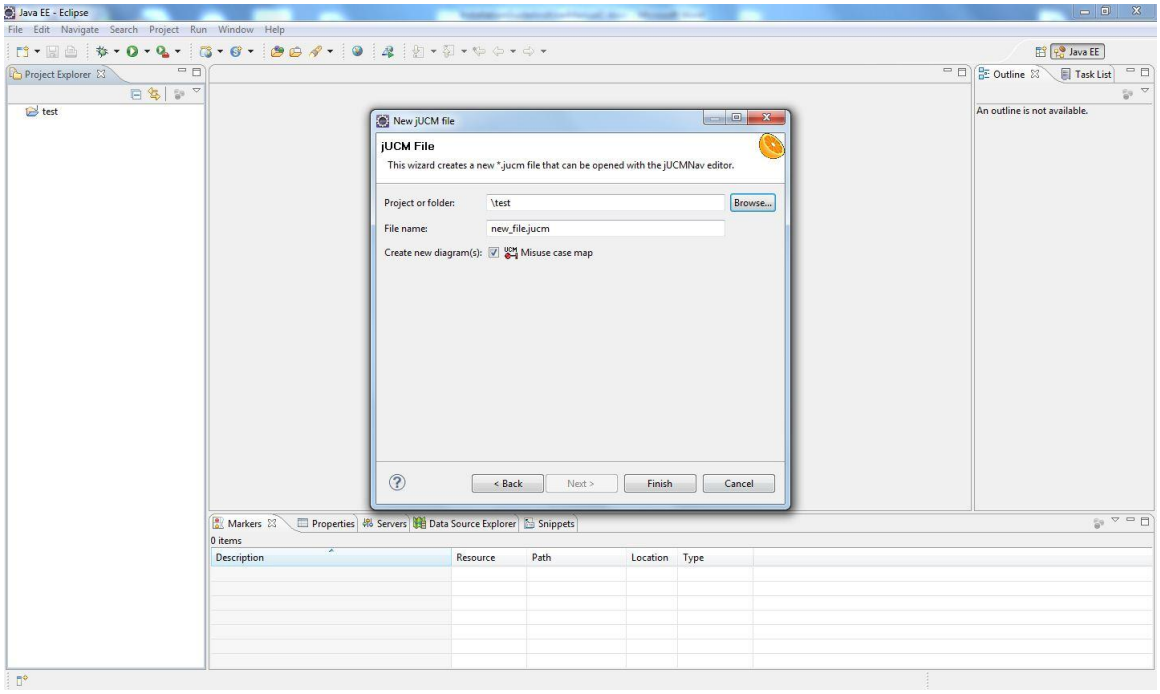


Fig. 12. New MUCM creation window

- Click on the finish button to start making diagrams. (Fig. 12)

- Use the left hand side palette to add different notations to your diagrams. (Fig. 13)

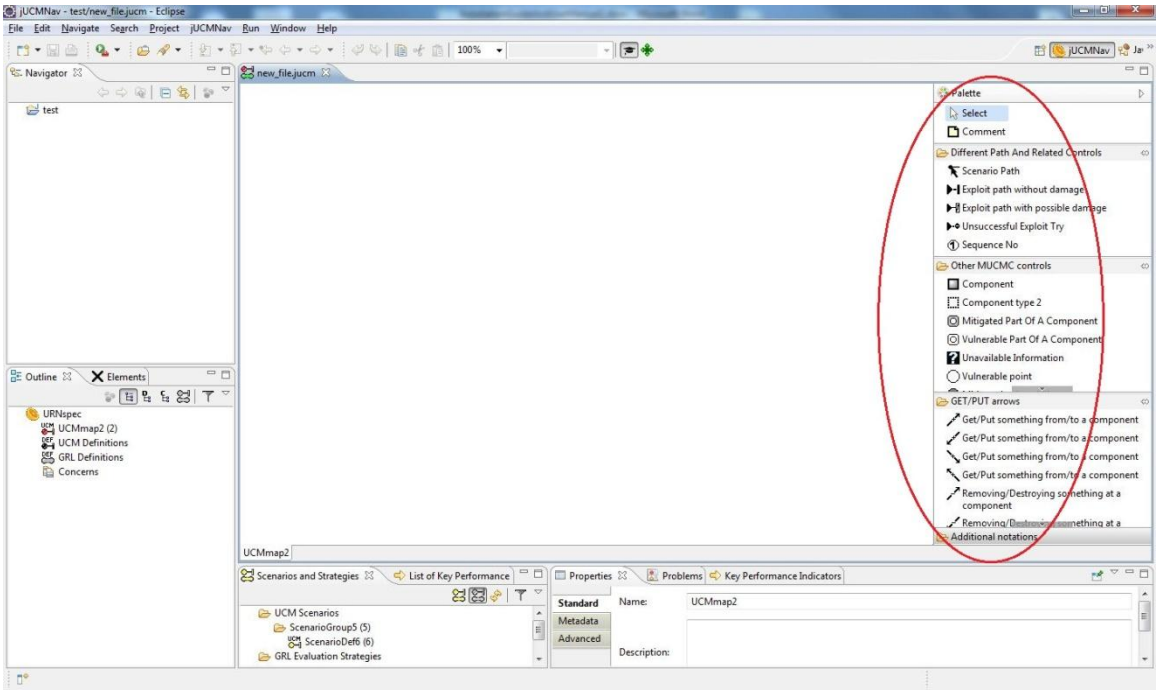


Fig. 13. Notation palette

3.2. Working with Notations

You can use different notations on the left side palette to make your misuse case map diagrams. Notations are categorized in different drawers based on their functionality. You can open and close drawers by clicking on the drawer's header caption. We discuss in what follows some important notations and give you some hint in working with the editor.

3.2.1. Different paths and empty points

Inside the path drawer you can find four different paths with different start points and end points.

- To add a path in your diagram, select your desired path in the drawer and click on the main screen.
- To extend a path, continue with additional clicks on the screen.
- When you are finished with a path and want to place another path, choose the "select" arrow and remove the focus from the old path by clicking on an empty place on the screen, select a new path from the drawer and repeat the previous steps.
- Every path consists of an end point, start point, linking edges and empty points. (Fig. 14) (please consider that the clouds are not part of the notations in the figure)

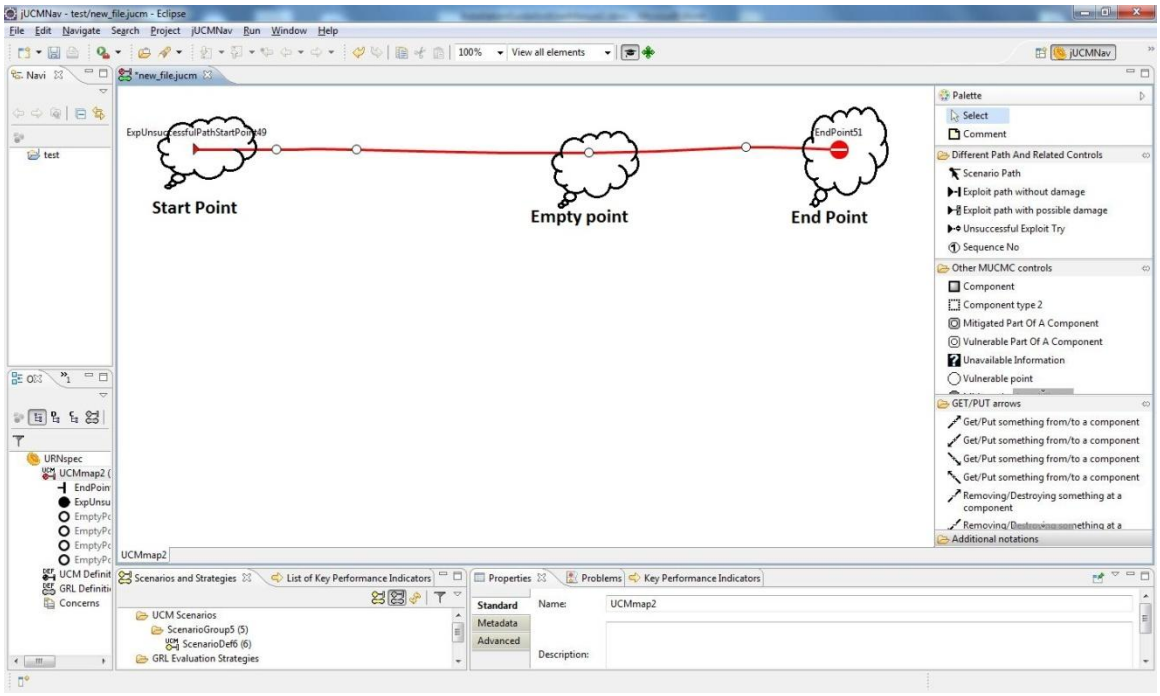


Fig. 14. A sample path in a MUCM diagram

- You can change the shape of your path by dragging the empty point on the screen
- For dragging the empty points, first choose "select" arrow to deselect the previous tool then select an empty point on the screen and drag it to a new place. (Fig. 15)

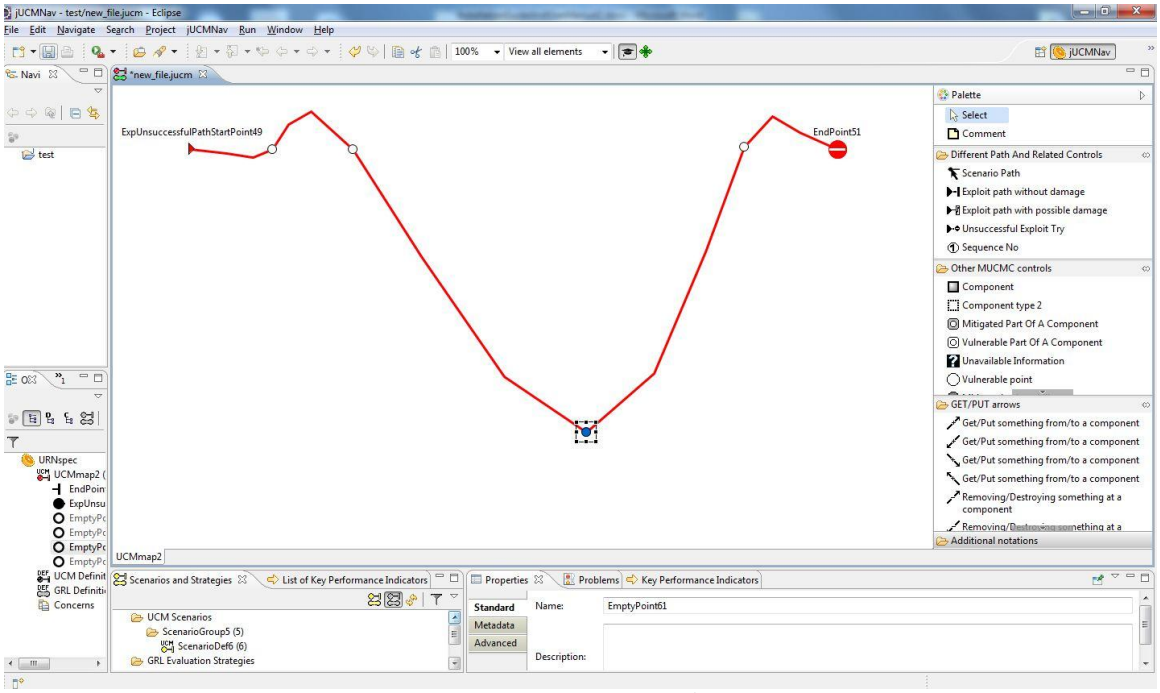


Fig. 15. Changing the shape of the path

- You can hide empty point whenever you want by selecting “Hide empty points” from upside combo box. (Fig. 16)

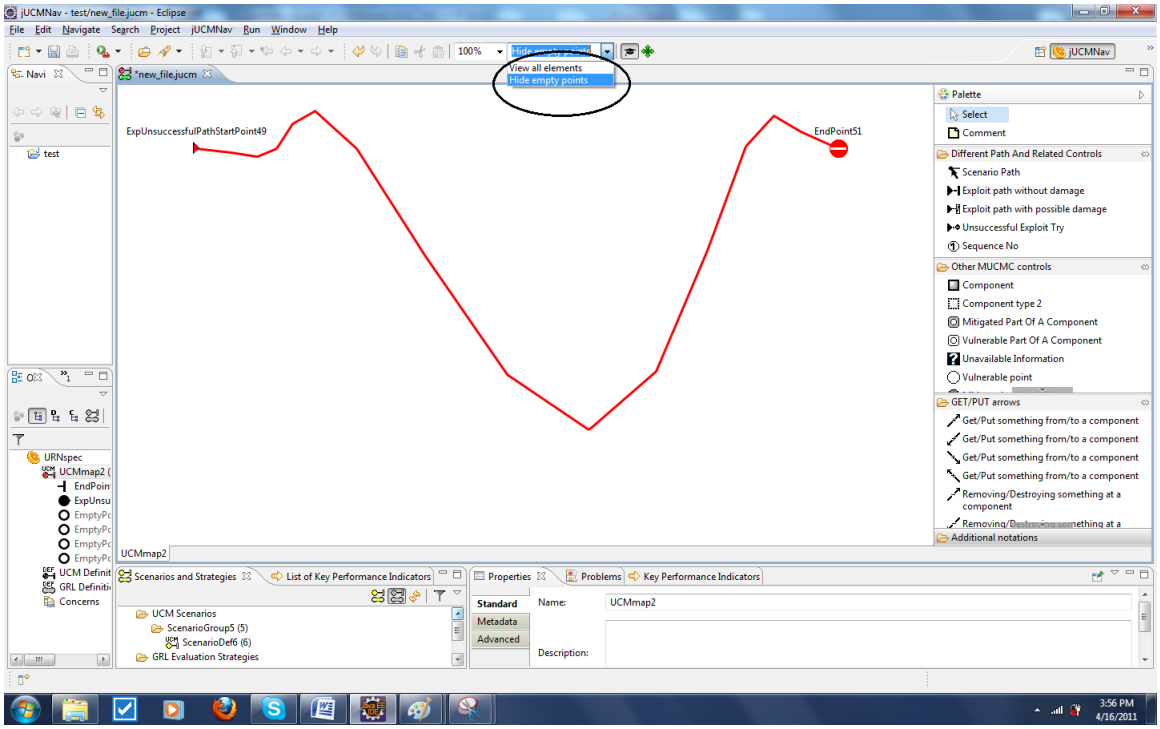


Fig. 16. Hiding the empty points

3.2.2. Additional Notations and vulnerable points

In “Additional notations” drawer you can find different vulnerable points notations which are the same as vulnerable points in the “Other MUCM controls” drawer. There is a major difference between these two categories of vulnerable points. You can place additional vulnerable points only on the empty points of the paths, and they will be attached to the target path. On the other hand, you can place the vulnerable points in “Other MUCM controls” drawer wherever you want without any restrictions.

3.2.3. Additional notation and “Anything point cut”

This notation is just allowed to be place on the empty points of the different paths like the vulnerable points in this category.

3.2.4. Binding notation together

You can bind the notations which are enclosed in a component to the parent component, by right clicking on the component and selecting the "Bind all enclosed elements" from the popup menu. After binding, all enclosed elements will be attached to the parent component and will be moved by moving the component. (Fig. 17)

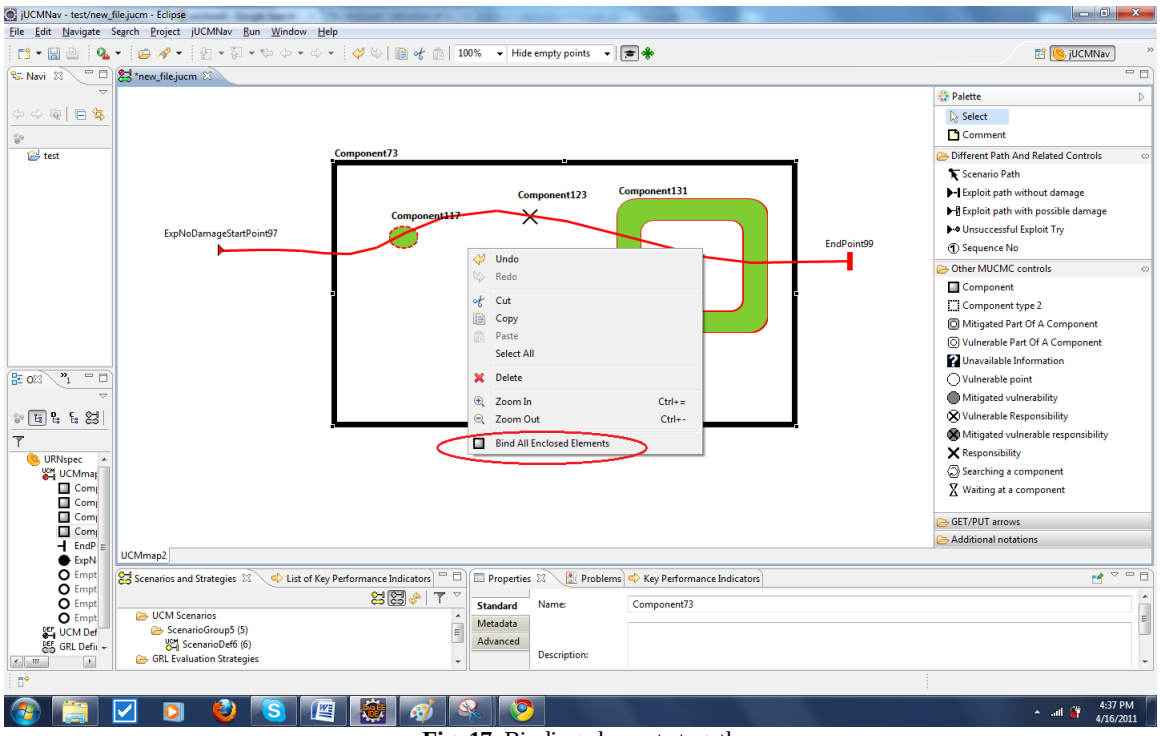


Fig. 17. Binding elements together

3.3. Printing Diagrams

For printing any MUCM diagrams, you have to export the diagram to a picture format like GIF or JPG. For exporting diagrams please go through the following steps.

- Right click on the MUCM file in the navigator panel and select the export from the popup menu.(Fig. 18)
- Choose "export UCM/GRL/URN" from the export window. (Fig. 19)
- Click on the next button and select "export individual diagrams" from the wizard. (Fig. 20)
- Click on next button and Select "JPEG Image" as a file type to export. (Fig. 21)
- Finish the export process by clicking on the finish button (Fig. 21)

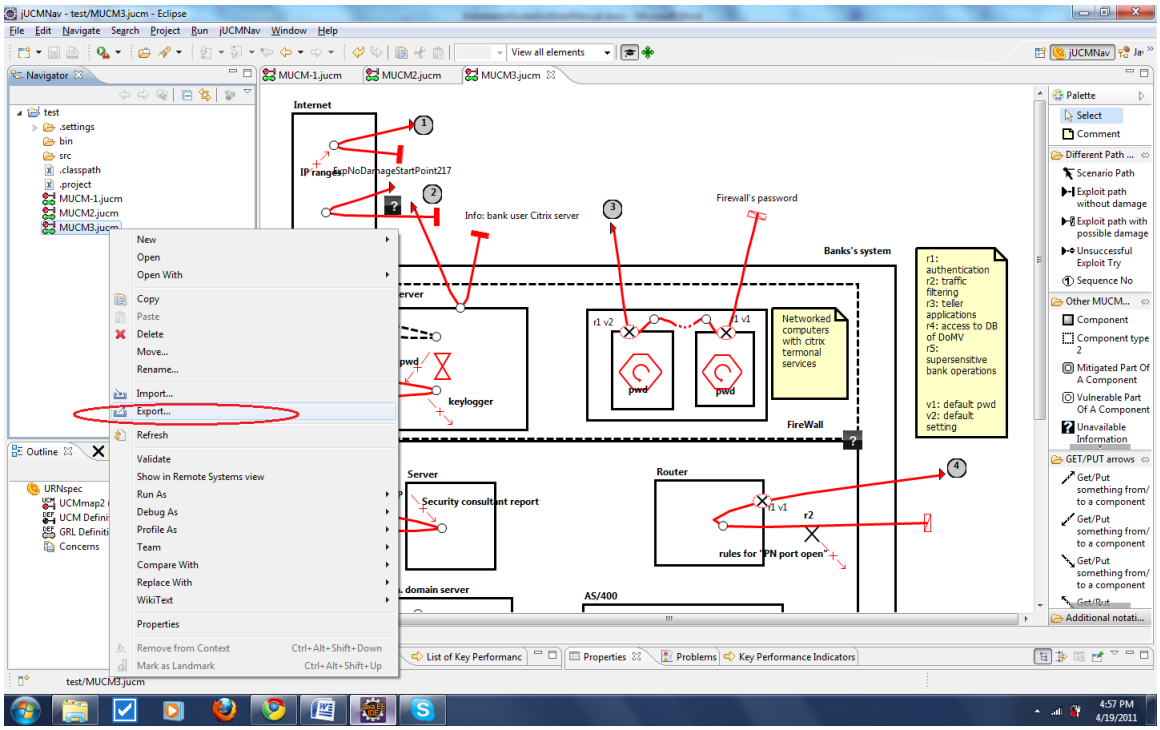


Fig. 18. Export menu

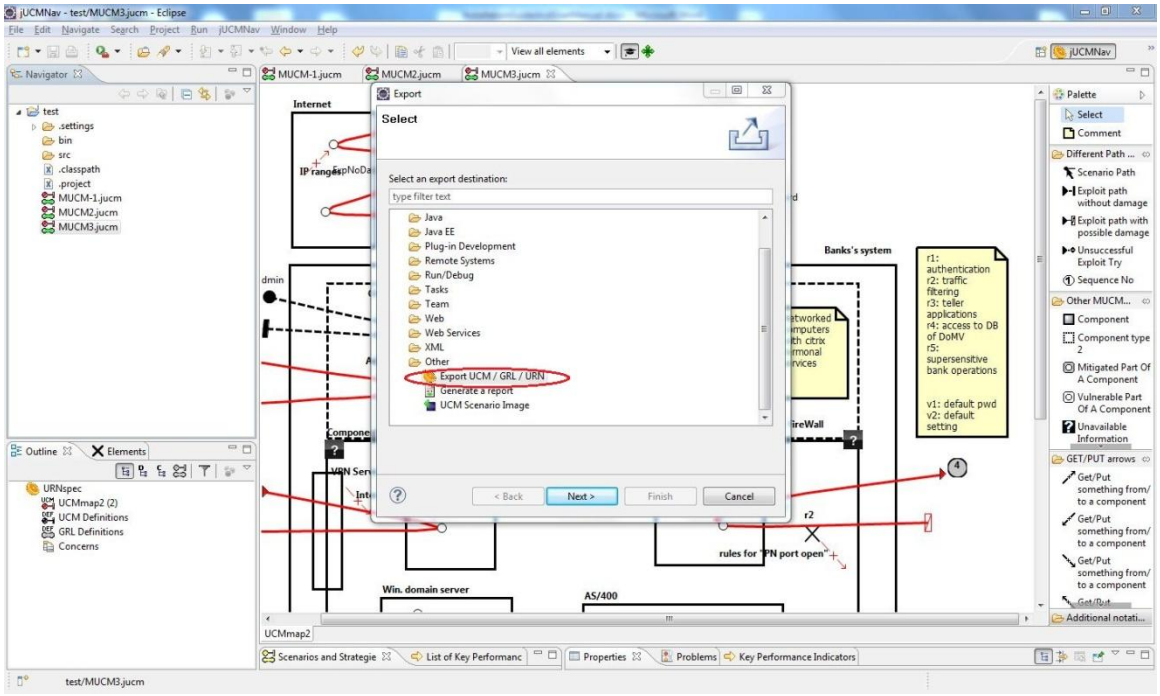


Fig. 19. Export Window

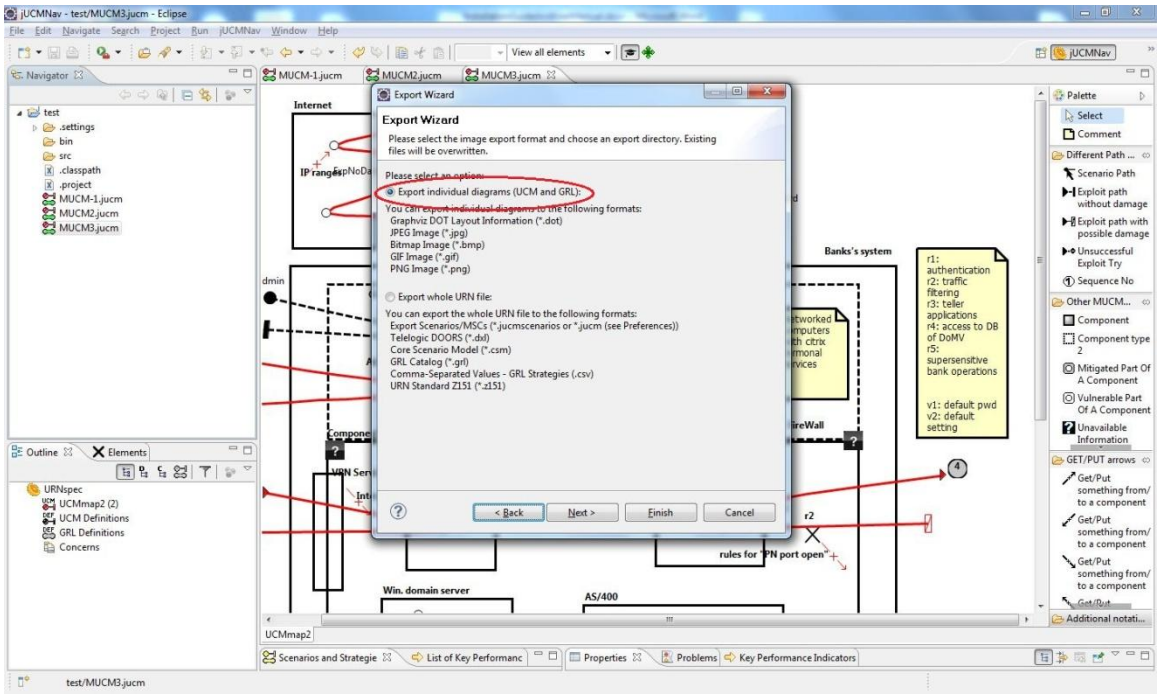


Fig. 20. Export wizard

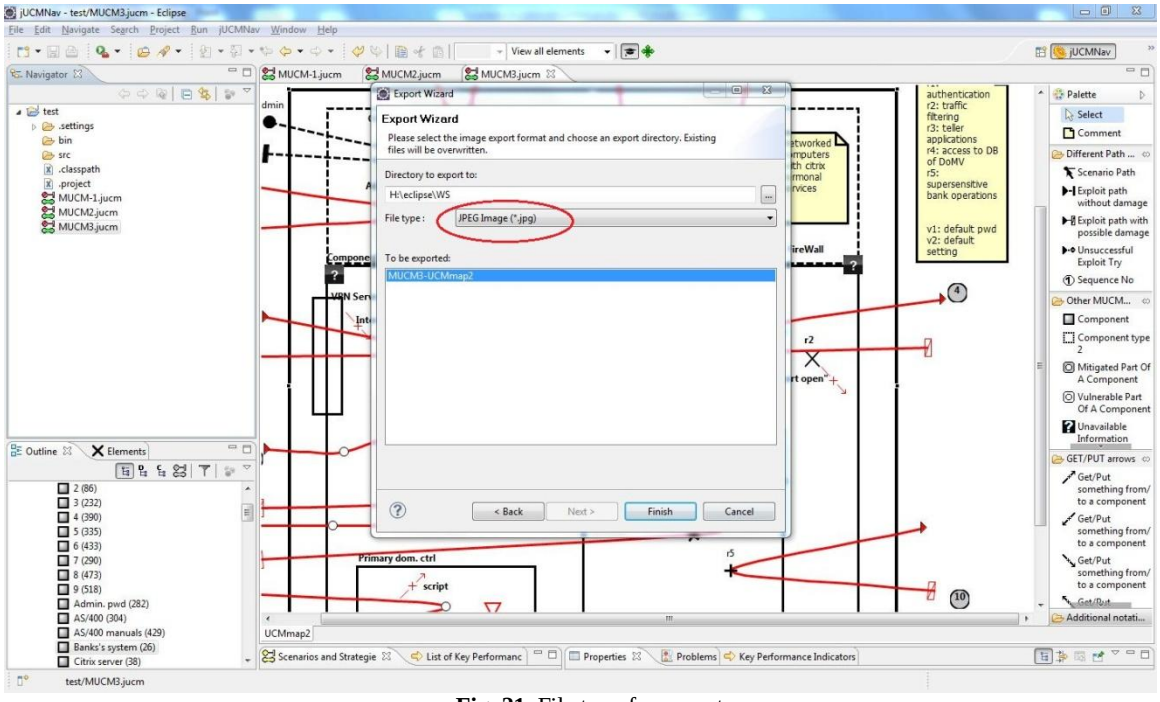


Fig. 21. File type for export

Appendix B: Pre experiment questionnaire

1. Participant ID:_____

2. Please, indicate your IT-relevant education level (i.e. none, bachelor student, finished bachelor study, master student, finished master study, PhD student, finished PhD, professor, other – please, explain it): _____

3. Please, indicate the number of years you have worked in IT. For part-time jobs, express as full-time equivalents, e.g., if you held a 50% IT job for 2 years, answer "1 year". _____ year(s)
4. Please, rate your expertise in the following areas where 1 is “Never heard of it” and 5 is “Expert”

Techniques and Methodology	**Never heard of it**	**Read about it**	**Tried it out**	**Used it a lot**	**Expert**
Requirement analysis of software systems	1	2	3	4	5
Designing software systems	1	2	3	4	5
Coding software systems	1	2	3	4	5
Testing software systems	1	2	3	4	5
Modeling software systems	1	2	3	4	5
Security analysis of software systems	1	2	3	4	5
Use cases	1	2	3	4	5
Misuse cases	1	2	3	4	5
Use case maps	1	2	3	4	5
Misuse case maps	1	2	3	4	5
The general purpose drawing tool of your choice for the experiment.	1	2	3	4	5

Appendix C: post-task questionnaire

	Strongly ***disagree***				*Strongly* ***agree***
Using the jMUCMNav editor by drawing the MUCM helped me focus better on the elicitation tasks than using the other alternative drawing tool.	1	2	3	4	5
It was easier to draw a MUCM using the alternative drawing tool than using the jMUCMNav editor.	1	2	3	4	5
If I had to draw a MUCM about a simple intrusion case then I would prefer to use my selected alternative drawing tool instead of the jMUCMNav editor.	1	2	3	4	5
The MUCM drawn by the jMUCMNav editor is better structured than the MUCM drawn by the other alternative drawing tool.	1	2	3	4	5
If I had to draw a MUCM about a complex intrusion case then I would prefer to use the jMUCMNav editor instead of my selected alternative drawing tool.	1	2	3	4	5
If I had to understand an intrusion case visualized as a MUCM, I would prefer to look at a version drawn by the jMUCMNav editor instead of a version drawn by the other alternative drawing tool.	1	2	3	4	5
Using the alternative drawing tool to draw the MUCM helped me to understand the specific intrusion case better than using the jMUCMNav editor.	1	2	3	4	5
It was easier to change or correct a MUCM using the jMUCMNav editor than using the other alternative tool.	1	2	3	4	5
If I had to develop a MUCM on spot in a brain storming meeting, I would prefer to draw it by the alternative drawing tool instead of using the jMUCMNav editor.	1	2	3	4	5
Using the jMUCMNav editor added unnecessary complexity to the drawing compared to the use of the other alternative general purpose drawing tool.	1	2	3	4	5
It was easier to work with the MUCM drawn by the alternative drawing tool than with the MUCM drawn by the editor.	1	2	3	4	5
It was easier to apply the MUCM notation when using the jMUCMNav editor than using the other alternative drawing tool.	1	2	3	4	5

Appendix D: “Bank hack” intrusion case

When Gabriel was surfing the Internet, it brought up details about IP addresses of the Dixie bank. He followed the trail, and discovered that it was not a small-town bank he had stumbled onto. It was a bank with extensive national and international ties. Even more interesting, he also found that one of the bank's servers was running Citrix MetaFrame, which is server software that allows a user to remotely access his or her workstation. A light bulb went on because of something that Gabriel and a friend had realized from their earlier hacking experiences.

This friend and I had discovered that most of the systems running Citrix services don't have good passwords. They deliver them already enabled, but leave the end user without a password.

Gabriel went to work with a port scanner, a hacker tool (or auditing tool, depending on the user’s intent) that scans other networked computers to identify open ports. He was looking specifically for any systems with port 1494 open, because that’s the port used to remotely access the Citrix terminal services. Therefore, any system with port 1494 open was a potential system he could successfully “own”.

Each time he found one, he'd search every file on the computer for the word password. It's like panning for gold. Much of the time, you come up empty-handed, but occasionally you discover a nugget. In this case, a nugget might be a reminder that someone had stuck in a file, maybe reading something like, "administrator password for mail2 is 'happyday. ' "

In time, he found the password to the bank's firewall. He tried connecting to a router, knowing that some common routers come with a default password of "admin" or "administrator," and that many people - not just clueless homeowners but, too often, even IT support professionals - deploy a new unit without any thought of changing the default password. And, in fact, that's what Gabriel found here - a router with a default password.

Once he had gained access, he added a firewall rule, allowing incoming connections to port 1723 - the port used for Microsoft's Virtual Private Network (VPN) services, designed to allow secure connectivity to the corporate network for authorized users. After he had successfully authenticated to the VPN service, his computer was assigned an IP address on the bank's internal network. Fortunately for him, the network was "flat," meaning that all systems were accessible on a single network segment, so that hacking into the one machine had given him access to other computer systems on the same network.

The hack into the bank, Gabriel says, was so easy it was "pretty dumb." The bank had brought in a team of security consultants, who provided a report when they left. Gabriel discovered the confidential report stored on a server. It included a list of all the security vulnerabilities that the team had found - providing a handy blueprint of how to exploit the rest of the network. Gabriel felt free to install Spy Lantern Keylogger, his favorite in the category primarily because of the program's unique ability to record information simultaneously from any number of people logging in to the Citrix server. With this installed, Gabriel waited until an administrator logged in, and "snarfed" his password. Armed with the right passwords, it was easy to gain full control of the system.

Appendix E: “Penetration test” intrusion case

With their get-out-of-jail-free cards, the 10pht team members could be as aggressive as they wanted, even "noisy" - meaning carrying out activities that could call attention to themselves, something a pen tester usually avoids. But they still hoped to remain invisible. "It's cooler to get all this information and then at the end know they had not detected you. You're always trying for that," says Carlos.

Newton's Web server was running the popular server software called Apache. The first vulnerability that Mudge had found was Newton's Checkpoint Firewall-1 had a hidden default configuration (rule) to allow in packets with a source UDP (User Data Protocol) or TCP (Transmission Control Protocol) port of 53 to almost all the high ports above 1023. His first thought was to attempt to mount off their exported file systems using NFS (Network File System), but quickly realized that the firewall had a rule blocking access to NFS daemon (port 2049).

Although the common system services were blocked, Mudge knew of an undocumented feature of the Solaris operating system that bound rpcbind (the portmapper) to a port above 32770. The portmapper assigns dynamic port numbers for certain programs. Through the portmapper, he was able to find the dynamic port that was assigned to the mount daemon (mountd) service. Depending on the format of the request, Mudge says,

"the mount daemon will also field Network File System requests because it uses the same code. I got the mount daemon from the portmapper, and then I went up to the mount daemon with my NFS request."

Using a program called nfsshell, he was able to remotely mount the target system's file system and download the entire exported file systems.

Mudge also found that target server was vulnerable to the ubiquitous PHF hole. He was able to trick the PHF CGI script to execute arbitrary commands by passing the Unicode string for a newline character followed by the shell command to run. Looking around the system using PHF, he realized that the Apache server process was running under the "nobody" account. Mudge was pleased to see that the systems administrators had "locked down the box" - that is, secured the computer system - which is exactly what should be done if the server is connected to an untrusted network like the Internet. He searched for files and directories, hoping to find one that was writable. Upon further examination, he noticed that the Apache configuration file (httpd.conf) was also owned by the "nobody" account. This mistake meant that he had the ability to overwrite the contents of the httpd.conf file.

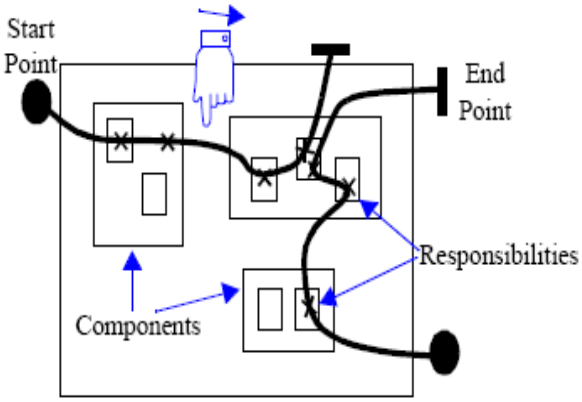
His strategy was to change the Apache configuration file so the next time Apache was restarted, the server would run with the privileges of the root account. Because the system administrator had apparently changed the ownership of the files in the "conf” directory to "nobody," Mudge was able to edit httpd.conf, so the next time Apache was started, it would run as root. (This vulnerability in the then-current version of Apache has since been corrected.)

Because his changes would not go into effect until the next Apache was restarted, he had to sit back and wait. Luckily for them, there was a blackout in the city where the company was located. That would mean the server would shut down. When the city power was restored, the system would reboot. Once the server rebooted, Mudge was able to execute commands as the root through the same PHF vulnerability;

While those commands had previously been executed under the context of the "nobody" account, now Apache was running as root. With the ability to execute commands as root, it was easy to gain full control of the system.

Appendix F: Misuse case maps introduction

A **Use Case Map (UCM)** depicts an overview of a system's architecture and some chosen behavior scenarios together. A UCM consists of components which can have some responsibilities connected to them which can be utilized by scenario paths in a causal chain. The basic notation and an explanation can be seen in Fig. 1.



Imagine tracing a path through a system of objects to explain a causal sequence, leaving behind a visual signature. Use Case Maps capture such sequences. They are composed of:

- **start points** (filled circles representing pre-conditions or triggering causes)
- causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- and **end points** (bars representing post-conditions or resulting effects).

The responsibilities can be bound to **components**, which are the entities or objects composing the system.

Fig 1. Basic notation and interpretation of UCMs

Misuse case maps (MUCM) use the basic idea of use case maps to depict and analyze complex intrusion scenarios. They depict involved architectural elements with their vulnerabilities and the order in which they were misused.

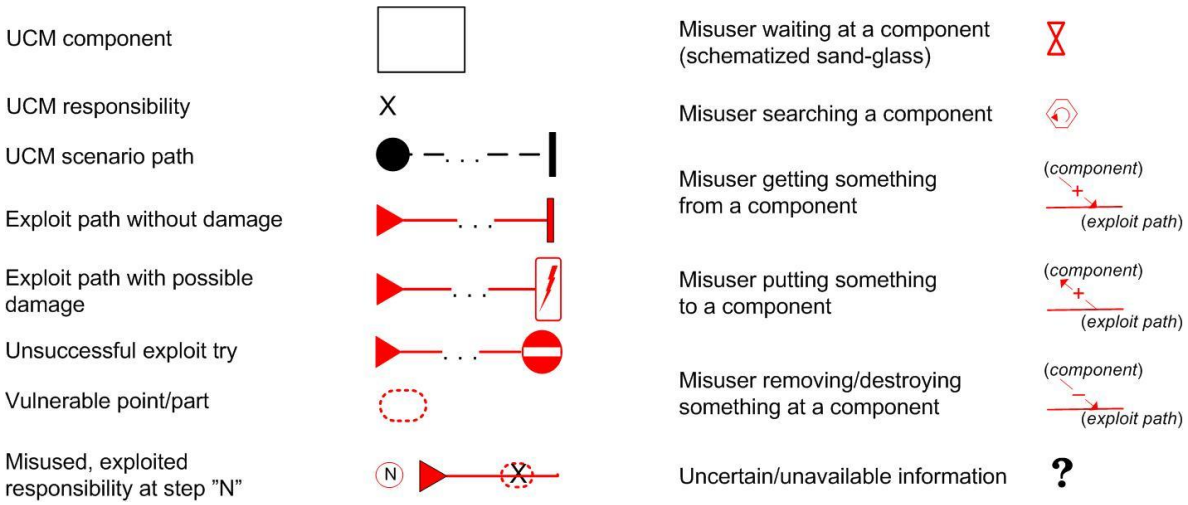


Fig 2. MUCM's basic notation and its interpretation

The MUCM notation (see Fig. 2.) is an extension of UCM basic notation. The UCM scenario path was changed from continuous line to dashed line for differentiating. Intrusions are represented by one or more exploit paths (continuous line) cutting through vulnerable parts of the system. An exploit path starts with a triangle. When a vulnerability is exploited then it ends in a lightning symbol. When an exploit is stopped by an appropriate countermeasure then it ends in a "no entry" symbol. In any other cases (e.g. when the attacker collects information or more happenings are combined in one exploit path) then it ends in a bar. The exploit paths are numbered showing the order of the intrusion steps. These steps are mostly causally related; in other words, they build on the result of a previous step. Only those responsibilities of the components are shown which are relevant for the intrusion.

The system may have vulnerable points (such as authentication responsibility) or parts (such as components without up-to-date security patches) which are suspect to threats. Misuses are depicted by the exploit path's crossing of a vulnerable point or part. Get, put ('+') and remove ('-') arrows can be used to show how an exploit path interacts with a component. An example of a put arrow is when the attacker ('arrow starting from the exploit path') installs ('+') a sniffer program on one of the servers ('arrow ends inside the component'). An example involving the get arrow is when the attacker gets a copy of a file. An example of the remove arrow is when the attacker deletes the traces of the intrusion. The simplified sand-glass symbolizes that the hacker has to wait for an event to occur in order to continue with the intrusion. The hexagon cell with the arrow symbol indicates searching in a component and the question-mark denotes still unclear details. The responsibilities and vulnerabilities have simple labels referencing their descriptions listed in a box outside of the MUCM. In addition, labels might also appear at other symbols to be more specific of a regarded issue, for example at misuser transactions ("get, put, remove" arrows).

Fig. 3 presents a simple example for the notation showing two components: Server X in System Y. System Y consists of more components which are neglected here in order to preserve simplicity. The attacker aim was to take control over System Y. He managed to learn more about a user of System Y in a previous step, i.e. her user identifier and birth date from facebook. In the next step, he tried to access Server X by using the user's facebook identifier and her birth date as password where he succeeded (r1 & v1). After finding out that the user privileges are inadequately set at Server X (r2 & v2), he was able to install a keylogger at the server ("put arrow"). His aim was to snatch the administrator's password which he hoped to work for other servers as well. The simplified hourglass symbolizes that the hacker had to wait for an administrator to log in. When it happened, the hacker got the administrator's password ("get arrow") through the keylogger and could continue his intrusion.

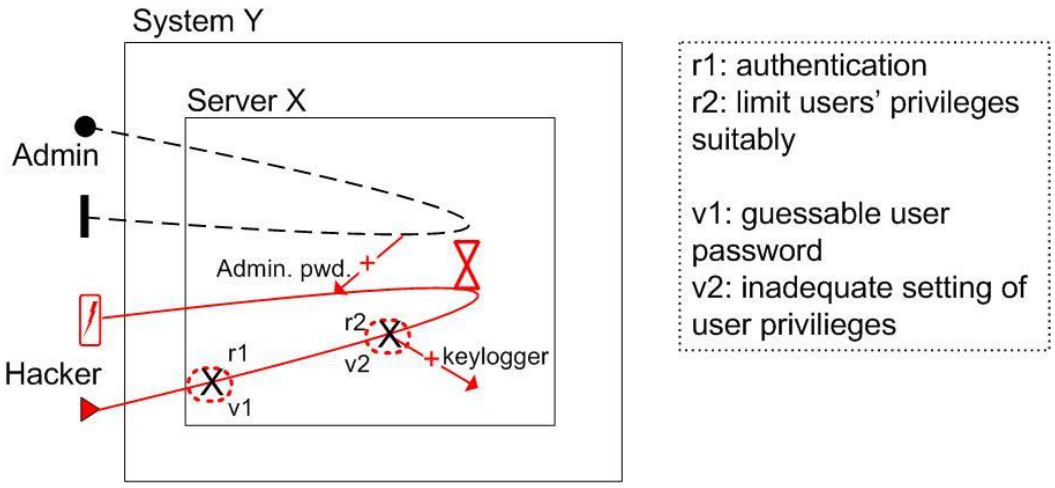


Fig. 3. A simple MUCM example