Security Assurance Case Diagram Design Tool

Design Document

Dec1718

Advisor and Client:

Dr. Othmane

Team Member:
John Koehn: Concept holder
Brandon Huegli: Team Lead
Chiakang Hung: communication
Jordan Lawrence: Webmaster

1718dec@iastate.edu
http://dec1718.sd.ece.iastate.edu
Revision: 21 April 2017

Contents

1 Introduction

1.1 Project statement

We are developing an eclipse plugin that will be used to design and manage security assurance cases. Users will be able to create, edit, and save/load assurance case diagrams, which look similar to a UML diagram in structure. Nodes of the diagrams represent specific claims or evidence, and will be tied to portions of the code and unit tests for the project the plugin is loaded into. Claims or evidence that are changed or are no longer valid will be marked visually for the user.

1.2 Purpose

The plugin will make the development and management of assurance case diagrams much faster and more intuitive. As a result, users will save money and time developing cases. Additionally, because the plugin will track validity of claims, it will ensure accuracy of the overall claim, and improve security.

1.3 Goals

1. Create an easily accessible Eclipse plugin that any developer could download
2. The GUI will be easy and intuitive to use for the developer. They should be easily able to link code in different projects that meet different security cases and be able to develop test to verify they meet them.
3. The plugin will give visual feedback to the developer if they are no longer meeting a security case
4. The plugin will actually be helpful to developers and speed up the development process
5. The plugin has documentation that a developer could understand to learn how it works and be able to use successful
6. Help Dr. Othmane further his research and be able to use the plugin in the future.

2 Deliverables

- A plugin importable into any existing eclipse project.
- A GUI for creating and modifying assurance case diagrams.
- A system where diagram nodes are tied to code snippets within the project, as well as unit tests.

3 Design

As the primary component from a user's perspective is the GUI, it was important to select an appropriate framework for developing the interface, keeping in mind the requirements of the project. We then decided which was a more promising candidate, and selected Graphiti to proceed with development.

3.1 System specifications

3.1.1 Non-functional

1. Visually pleasing and professional interface

2. Intuitive and easy to use.

3.1.2 Functional

1. Ability to add/remove graphical elements to assurance case diagram

2. Multiple graphical elements to choose from
    a. Case claim
    b. Evidence
    c. Edges

3. User interaction with elements
    a. Move elements around the chart
    b. Edit text in the element
    c. Draw connections between elements

4. Open submenu for each node to connect underlying code or tests to elements
    a. Open currently linked code sections
    b. Select new code sections to link
    c. Update diagram node in response to code changes appropriately

5. Visually alert user when claims/evidence nodes are no longer valid.
    a. Propagate invalid status to parents.
    b. Failed tests change element color.
    c. Changed, potentially vulnerable, code changes element color.

6. Save/Load diagram as part of the Eclipse project

7. Import and export diagram files into the project
    a. Use human readable formats, so diagram files can be edited directly.

3.1.3 Standards

We will be following the IEEE code of ethics for this project. Our tool is meant to help developers make more secure software. If our tool has a bug that results in it not notifying a developer that a security case is no longer being meet then we would be compromising a software's security which could cause damage.

The plugin will also meet established conventions on the design of safety assurance cases as specified by Carnegie Mellon University, and governmental agencies such as NASA and the FDA. (Goodenough, et. al., Ponsard et. al., Carnegie Mellon University)

3.2 PROPOSED DESIGN/METHOD

The GUI of the project, and the overall plugin structure will be implemented using the Eclipse Graphiti framework, which handles lower level interactions, and allows creation of the necessary diagram components. The supporting features, as well as Graphiti itself use Java.

3.3 DESIGN ANALYSIS

At the start of the semester, we experimented with a variety of GUI frameworks to use, as we had decided Java Swing/Fx was not sufficient for our project. After evaluating the options, we selected the Eclipse Graphiti framework for development. From there, we decided the first phase of development should be the plugin's GUI. The primary requirements and features of the GUI will be completed this semester.

Next semester, we will begin on "phase two" of the plugin, and begin adding desired supporting features, like the ability to tie code and test cases to diagram elements, and other features specified with Dr. Othmane's input.

4 Testing/Development

4.1 INTERFACE SPECIFICATIONS

In our project we are interfacing with Eclipse to create a plugin for Eclipse. We also need to be able to interface with other Eclipse Projects to do analysis on the code of other projects.

Additionally, while the plugin will not do so directly, testing of our product will be done in conjunction with existing security software, including Zen Cart, along with existing documentation and specifications for assurance cases.

We are examining source code from the eclipse plugin project EditBox[1] (see references "Other") to see how we can interface our plugin with source code of other projects. Once we understand how EditBox does it, we will take the methodology used and implement it into our code.

4.2 HARDWARE/SOFTWARE

JGraphx*[1]	Would be able to meet the requirements of the project. Would be easy to begin development with.	Built on Java Swing. Not eclipse specific. Dated aesthetic.	We decided that while this framework was a candidate as it met our requirements, it was not the best option available.
Eclipse Graphiti*[2]	Built for making eclipse plug ins. Aesthetically the most pleasing. Acceptable documentation and tutorials.	Team lacks experience with plug in development.	The team has chosen Graphiti for development of the project.
JGrapht*[3]	Would've been likely able to meet requirements in	Built on Java Swing.	This framework was not considered a

	most ways.	Not intended for the exact style of diagram necessary.	candidate, as it did not match our needs as closely as other options.
Graphviz*[4]	Able to draw diagrams needed.	Poor available information. Doesn't appear to be built for Java or Eclipse.	As with JGrapht, this library was not considered a candidate.

* Information taken from respective websites, see references "GUI Libraries."

4.3 Process

For each framework, we primarily based our decisions on the available documentation, and any provided examples. Many provided images or sample projects of what could be done with the framework, and described how they could be used. Based on those, we decided how closely the framework aligned with our requirements.

The plugin and GUI will be tested by using existing assurance cases as a reference, and ensuring we can replicate them. Then, we will test it using existing software, such as ZenCart.

5 Results

In the testing phase, we experimented with a lot of different GUIs. JGraphx and JGrapht are both outdated, complex and hard to understand. Meanwhile Graphviz lacked documentation and didn't have as many features and capabilities as we would have liked. However, Graphviz was considered a fallback in case Graphiti failed.

Graphiti in our testing proved to be very powerful. A lot of the issues we had with it initially was we didn't understand how to get the framework to run. However, after going through tutorials we started to understand it and got running examples. The tool is flexible and is able to accomplish a lot with less code than the other frameworks. Therefore we decided on Graphiti.

Our work with Graphiti as are graphic's API has been successful. We have been able to build our software to create intractable diagrams that we believe to be intuitive and meet the needs of complex software projects. As we become better

versed with the Graphiti library it is becoming easier to add more complex features into our tool. As we near completing our current list of features needed for the GUI, we have begun to devote resources to the next part of the project by researching how we can incorporate our tool with ZenCart code.

6 Conclusions

We have evaluated and selected a framework that will meet the requirements of the project, and are learning more about using it as we begin development of the project. We want to develop an eclipse plugin to manage security assurance case diagrams, and Graphiti is definitely the best option to meet these requirements, as it is essentially for diagram based user interfaces, and already takes the form of a plugin naturally. From there, the diagram editor has been completed based on our functional requirements for that portion of the plugin. Users are able to add, remove and edit nodes necessary to replicate existing assurance cases.

As work progresses, additional functional requirements will be implemented using standard Java, as Graphiti itself is built with Java. We are beginning to explore how we can interface our tool with code from other projects as our main GUI features get finished.

7 References

Assurance Case General Information:

Goodenough, John, Lipson, Howard, and Weinstock, Charles. "Arguing Security - Creating Security Assurance Cases." Carnegie Mellon University, 4 November 2017. https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/arguing-security-creating-security-assurance-cases. Accessed Feb. 2017

Ponsard C., Dallons G., Massonet P. (2016) Goal-Oriented Co-Engineering of Security and Safety Requirements in Cyber-Physical Systems. In: Skavhaug A., Guiochet J., Schoitsch E., Bitsch F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2016. Lecture Notes in Computer Science, vol 9923. Springer, Cham

Software Engineering Institute. Carnegie Mellon University, 2017, http://www.sei.cmu.edu/dependability/tools/assurancecase/. Accessed Feb. 2017.

Assurance Case Software:

Certware. Nasa, 2012. https://nasa.github.io/CertWare/. Accessed Feb. 2017

GUI Libraries:

[3]Barak Naveh. Jgrapht. http://jgrapht.org/. Accessed Feb. 2017

[2]*Graphiti.* The Eclipse Foundation. https://eclipse.org/graphiti/. Accessed Feb. 2017

[1]*Jgraphx*. https://github.com/jgraph/jgraphx. Accessed Feb. 2017

[4]John Ellson, Emden Gansner, Yifan Hu, Arif Bilgin, and Dwight Perry. *Graphviz*. http://www.graphviz.org/. Accessed Feb. 2017

Other:

[1]*EditBox*. https://github.com/Nodeclipse/EditBox. Accessed Apr. 2017