Extending snBench to Support A Graphical "Programming" Interface For a Sensor Network Tasking Language (STEP)

Ching Chang, Raymond Sweha, Panagiotis Papapetrou
Department of Computer Science
Boston University
{jching, remos, panagpap}@cs.bu.edu

Abstract

The purpose of this project is the creation of a graphical "programming" interface for a sensor network tasking language called STEP. The graphical interface allows the user to specify a program execution graphically from an extensible pallet of functionalities and save the results as a properly formatted STEP file. Moreover, the software is able to load a file in STEP format and convert it into the corresponding graphical representation. During both phases a type-checker is running on the background to ensure that both the graphical representation and the STEP file are syntactically correct. This project has been motivated by the Sensorium project at Boston University. In this technical report we present the basic features of the software, the process that has been followed during the design and implementation. Finally, we describe the approach used to test and validate our software.

1 Introduction

Sensor Task Execution Plan (STEP), is a sensor network tasking language used to develop applications on sensor networks. The purpose of this project is the development of a graphical "programming" interface for STEP. The graphical interface will allow the user to specify a program execution graphically from an extensible pallet of functionalities and save the results as a properly formatted STEP file. A STEP file is typically an XML formatted file that describes an execution plan for a sensor network application. Moreover, the software will be able to load a STEP file and convert it its corresponding graphical representation. During both phases a type-checker will be running on the background to ensure that both the graphical representation and

the XML file are syntactically correct.

This project has been named "STEPVIEW" and is motivated by the Sensorium project at Boston University [1]. Its ultimate goal is the production of a user-friendly software that will enable the user to write syntactically correct code in STEP just through a graphical interface and without getting into XML level coding details.

1.1 Team Organization

A successful implementation is a result of intense and focused team work. In this section we give an outline of the division of responsibilities among the team members along with a schedule of tasks until the submission date. Moreover, we had weekly meetings regarding our progress. Each meeting was divided into three parts: (1) touch-base from the last meeting, where we would discuss the progress with regard to the milestones set at the previous meeting, (2) general discussion, where each member would talk about her part, give new ideas and identify problems with the current implementation, (3) new milestones, where the milestones for the next meeting would be set. Also note that for the efficient collaboration we used CVS, which helped in the organization of the code and synchronization of the team work.

As of our first meeting in February 8th the responsibilities of the three team members have been broken down as follows:

1. Panagiotis Papapetrou: (1) Requirements Specification, (2) Web-site Design and Maintenance, (3) External Documentation.

2. Ching Chang: (1) GUI Design, (2) Step-to-GUI Conversion, (3) GUI maintenance.

3. Raymond Sweha: (1) GUI Design, (2) Type-Checker, (3) Internal Documentation.

In our second meeting in February 14th, we decided on the final schedule of the different parts of the project. This schedule is shown in the Gantt diagram available on our project web-site. The project has been broken down to eight major Tasks. Notice that Task 14 had been removed from our plan, since it is not part of the functional specifications of the project and it would require a great number of modifications to the data structure used to store the graphical representation. Also, we set four major milestones that have been completed successfully on time.

A Pert Diagram of the development of our project is available on our project web-site. Each node in the diagram corresponds to a project task and consists of two rows: the first one is the name of the task, the second shows the scheduled start and end dates, and the third one shows the dates of the actual start and finish of the task. As it can be seen from the diagram almost every task has been completed as scheduled and the delays were usually for one or two days. Notice that the STEP-to-GUI conversion exceeded the schedule by four days, whereas the Type-checker was completed five days ahead of schedule. Also, it should be mentioned that according to our first requirements specification, our software was also planned to be able to recognize SNAFU-formatted files (Task 14 on the Gantt Diagram). This objective though was considered (both by us and the faculty) quite hard to implement with the current design. So, it was later omitted from the Pert diagram. However, we have intentionally put it on the Pert diagram to show that it was initially planned.

1.2 The Waterfall Model

After examining various generic software process models for the development of our project, we agreed to use the waterfall model [2, 3]. Its basic characteristic is that it is based on separate and distinct phases of specification and development. The whole process is broken down to five phases: (1) Requirements analysis and definition, (2) System and software design, (3) Implementation and unit testing, (4) Integration and system testing, and (5) Operation and maintenance. Figure 1 gives a brief overview of how this model works. The main drawback of the waterfall model is the difficulty of accommodating any change after the process is underway. One phase has to be complete before proceeding to the next phase. Therefore, an inflexible partitioning of the project into distinct stages can make it difficult to respond to any changes of the customer requirements. As a result, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. Few business systems have stable requirements, so other models are preferred to this one. However, in our case the specifications were initially well-understood and the user-requirements were fully defined. Consequently, the use of the traditional waterfall model ended up being a good choice.

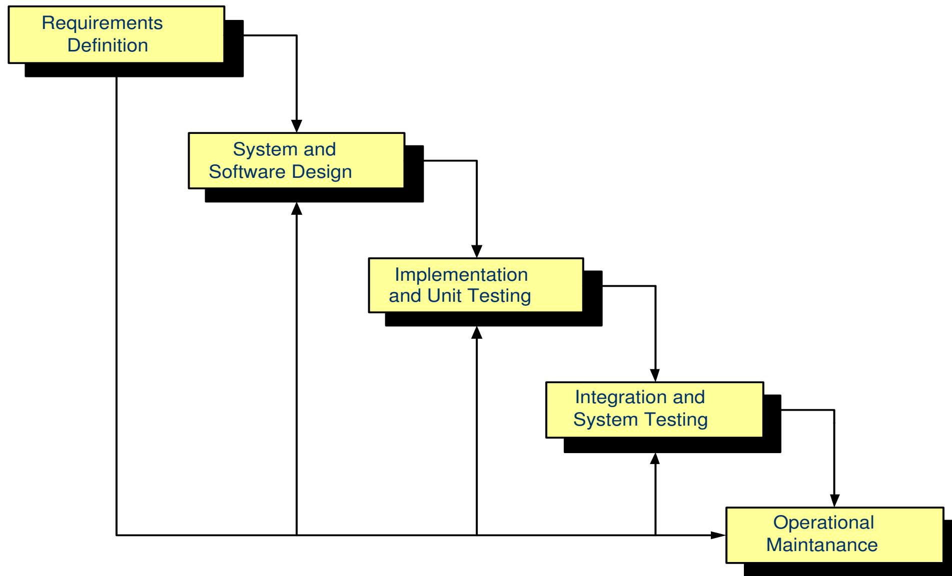

Figure 1: The Waterfall model.

2 Requirements Specification

This section presents the requirements specification of our project including the functional, interface and performance requirements. First we give a brief overview of the terms above followed by a more detailed analysis of the requirements. Our analysis is based on [3].

Functional requirements include the possible effects of a software system, in other words, what the system must accomplish. These specifications are determined by the project developers based on the user preferences and requirements. Interface requirements describe how the software interfaces with other software products or users for any given input or output. Performance requirements specify some performance issues of the software regarding speed and efficiency.

2.1 Functional Requirements

The functional specifications of our system are given below, in ranked order:

1. Design of a graphical "programming" interface for STEP: the graphical interface should allow the user to specify a program execution graphically. The GUI will provide the user with all the appropriate functions (in the form of buttons) to create a syntactically correct graph that can be successfully translated to a valid STEP file.

2. Recognize STEP-formatted execution files: the software should be able to load a program execution from a STEP formatted file and represent this execution graphically.

3. Type-checker: the software should have the ability to handle syntax errors online. The type-checker will be active during the program execution, detect syntax errors on demand and provide the user with helpful suggestions. The type-checker is not one of the main goals of this project. It has been implemented as an additional system component and it can validate any type of STEP node except for the "expression" STEP nodes. Thus, its main functionality is to assure that the designed graphical execution is a valid STEP file representation. For this module we followed the specifications given in the definition of "STEPNode" on the project's web-site.

4. Some other features: (1) the objects on the GUI are movable, (2) the user can zoom in and out by using the mouse wheel, (3) the properties of each node can be changed by right-clicking on the each node.

5. Interact with the snbench SXE environment: as soon as the graphical representation is completed, and the user has designed the STEP graph of the program she wishes to create, the representation is converted to the corresponding STEP file and then it is executed directly on snbench.

6. Recognize SNAFU-formatted execution files: the software should be able to load a program execution from a SNAFU formatted file and represent this execution graphically. This functional specification was originally planned as an extra component of the project, but was removed later on since it would require complete rearrangement of the current implementation.

2.2 Interface Requirements

This section describes how the software interfaces with other software products or users for input or output. Typically there are three main types of interfaces: user interfaces, hardware interfaces and communication interfaces. In this section we are going to focus on user interfaces.

The user interface requirements describe how the product interfaces with the user. In this case we have three types of interfaces:

1. GUI: the graphical user-interface is divided into three frames. The first one is located on the left side of the GUI and contains the set of buttons that offer the user all the functionalities needed to design the appropriate modules of a STEP program. The second is the area where the user designs the execution and where all the objects are placed. The third one shows the current version of the STEP file based on the objects that have been already designed and are linked to each other following the syntax rules of STEP. Also, a pop-up window is used to report any syntax errors during the graphical design of the execution.

2. INPUT: the input of the software can be of two different forms: (1) a STEP-formatted file given by the user which is assumed to be syntactically correct and corresponds to a valid STEP file, (2) a graphical execution of a STEP program.

3. OUTPUT: the output of the software depends on the input and can be of two different forms depending on the INPUT: (1) the graphical execution design of the STEP file that has been given as input, (2) a syntactically correct STEP file that has been created based on a graphical execution design.

2.3 Performance Requirements

In this section we address some performance issues and mainly focus on speed and correctness. First of all, we require our final product to be very accurate and produce STEP code with no errors. To assure this, we have enhanced a type-checker which, as described above, runs simultaneously with the application and evaluates the current execution design both after each step and on demand. At each step, a function is called that enables the type-checker and validates the current action. If there is a syntax error at the graphical representation of the code, the user is informed in two ways: first the validation window shows a message regarding the syntax error and at the same time the corresponding nodes or edges on the graph of the execution are colored differently showing that there is something wrong.

3 Package Organization

In this Section we give a precise description of the organization of our package into directories and Java source files. We also provide the UML class diagram of our software that gives a better view of the system and the way each module interacts with the others.

Our package consists of the following classes:

1. STEPGraph: is the generic class that describes the data structure that holds all the nodes and links created in the system. It contains all the appropriate functions and features needed to maintain the graphical representation efficiently.

2. NodeObject: is the class that describes each STEP node object in the system. It is the basic component of STEPGraph, since the STEPGraph is used to create, hold and delete all the NodeObjects of the system.

3. LinkObject: is the class that describes a link between two NodeObjects.

4. NodeOption: is the class that contains methods to change the features of each NodeObject.

5. IDEStarter: contains all the basic functions to initialize, configure and maintain the application.

6. SGRightClickPopup: is the class that describes the pop-up menus that change the GUI properties.

7. ValueOption: changes the ID values of the STEP nodes.

The way the above classes interact with each other is shown in Figure 2 that shows the class diagram (in UML) of our package.

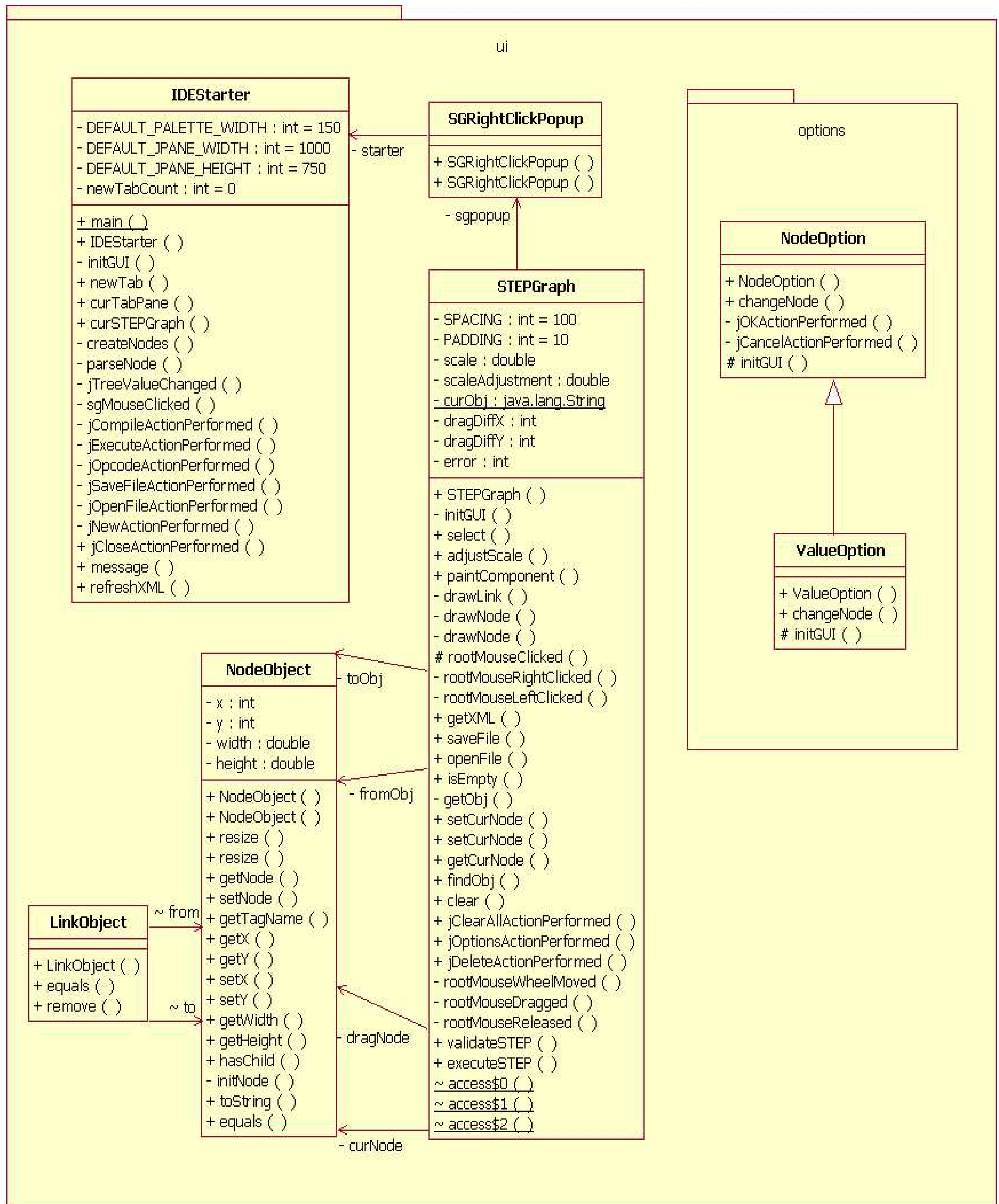

Figure 2: The UML class Diagram of our software.

4 Package Activation

This Section provides a clear presentation of how to activate and use our package. The code has been provided in the zip file called “source.zip”. To activate the code you need to follow three steps:

1. Unzip the file and save it under any folder.
2. Open Eclipse and create a new Project called “snbench”.
3. Add the unzipped files into “snbench”.
4. Open and run the file called “IDEStarter.java” which is under the following path: “snbench/ide/ui”.

To draw the graph the user can select any of the STEP node types included in the menu on the left. After two nodes are drawn, the user can draw a line to connect them depending on their types. At any point you can see the STEP file resulting from the current graphical representation. In case of an erroneous design, the corresponding STEP code is not generated. The type-checker can be used at any time by pressing the “Compile” button. This will type-check the current graph and enure that it corresponds to a valid STEP file. In case of any errors, they will be reported at the area designed at the bottom of the design pallet.

Moreover, our software can interact with snbench directly and actually run the STEP programs that are created graphically. To do this, the server needs to be activated before the software is started up. This can be easily done by running the following command in Unix: “java sxe.Server http://localhost:8080 NONE”. Then, after creating the desired execution graph and compiling it, we can run the resulting STEP file by pressing the “Execute” button.

The reverse process is also supported by our system. The user can load any STEP file, using the “load” button, and the system will create the corresponding graphical execution. In Figure 3 we give a screen-shot of the system. As we can see, on the left we have a menu of all the possible STEP nodes that can be included in a STEP file, and on the right we have two frames. The upper frame provides an area to design the graph, and the syntax errors are reported on the bottom frame. Another screen-shot is given in Figure 4, where an exception is shown.

Figure 3: A screen-shot of “STEPVIEW”.

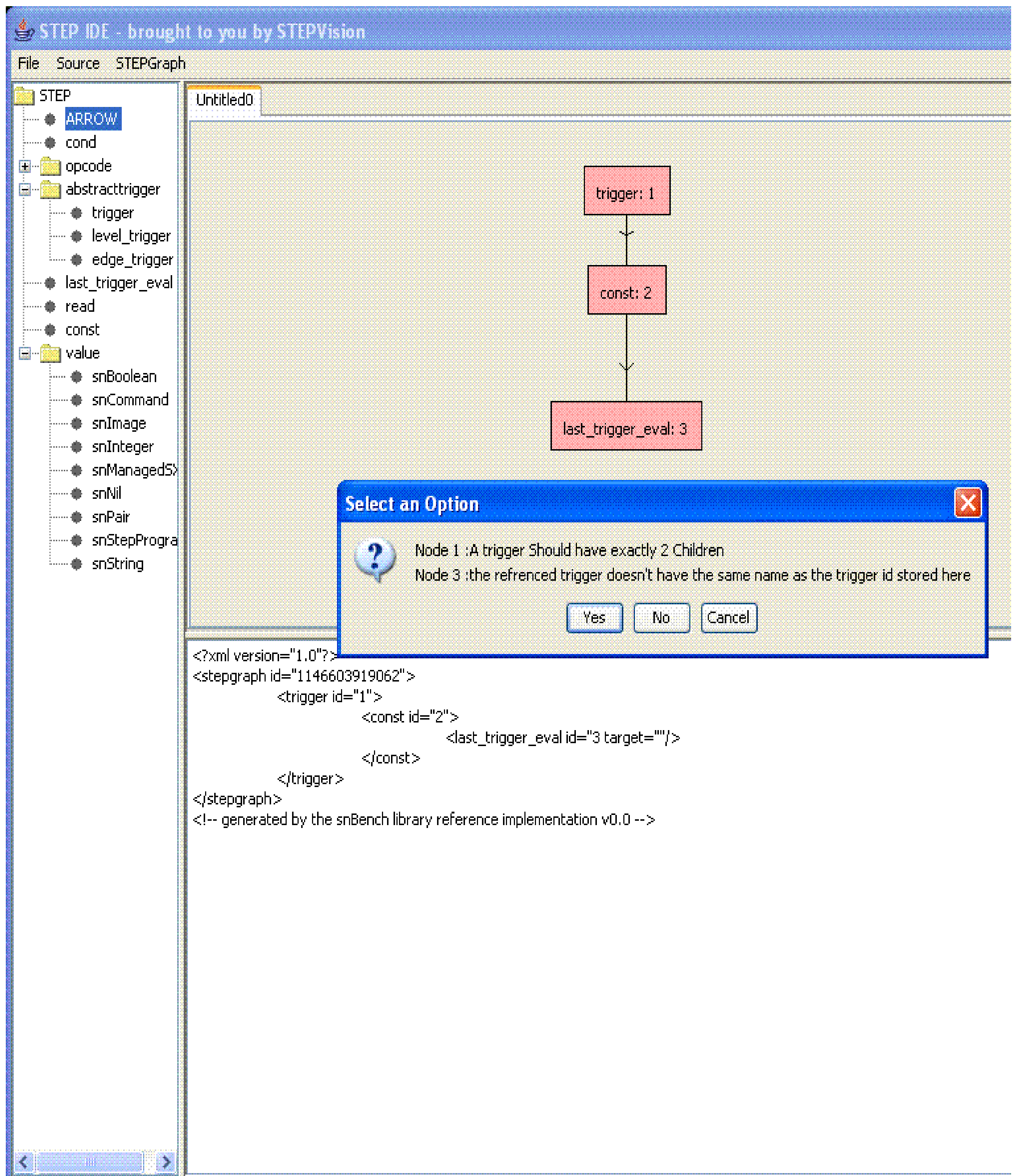

Figure 4: A screen-shot of “STEPVIEW” showing an exception notification.

5 Testing and Test Results

Testing our system was the last step of the development process. In this Section we describe the approach used to verify and validate our system. We further discuss how the tests have been selected and why they are sufficient.

We have tested our software on various input STEP files, that are included in the submitted zip file. To ensure the correctness of our code we tried to cover every possible case. First, we have tried all the different kinds of STEP nodes individually, to assure that the code works correctly for each one of them. Then we made combinations of two and three different types of STEP nodes. Moreover, we created several STEP file that contained all the different types of STEP nodes with random repetitions of some types. Finally, we tested our code on a set of STEP files that are available under the "testsuite" directory in Michael Ocean's code. The software produced the correct output for each input file.

Furthermore, we had to ensure that the software worked correctly for the reverse procedure, i.e. convert a given STEP graph on the GUI to its corresponding STEP file. This process was quite easy. For each input file described above we created the STEP graph and then converted each graph to its corresponding STEP file. In all cases, the new STEP file matched the original one, which indicated the correctness of our code.

Even though our test files covered every possible scenario, for completion, we finally tested our software on the three examples given on the project web-site: $http://cs{-}people.bu.edu/mocean/cs511$.

6 Reflection and Future Work

In this last Section we discuss the design and implementation techniques that distinguish this project as a success. We further identify and discuss some known limitations and propose directions for future work.

6.1 Reflection

"STEPVIEW" has been completed successfully and has met all our expectations and goals. We managed to create a user-friendly interface to graphically represent the execution of STEP files.

The interface provides the users with all the appropriate utilities and gives them a variety of menus and options to create the STEP execution graphs easily and efficiently. For our implementation we have used part of Michael Ocean's code, we have extended it by overriding some methods and improving their functionalities.

One problem of our implementation is the fact that it requires each STEP node to have its own unique id. In other words, our software does not allow any two STEP nodes to share the same id. Therefore, if our program is given an input STEP file that contains a duplicate id, an exception is raised and the STEP file is rejected. This problem could not be solved easily since the original implementation (Michael Ocean's) required the ids to be "static" and our code had to be compatible with the original code.

6.2 Our Contributions

Our main contributions include:

1. Build STEP programs using GUI.
2. Save the STEPGraph in a STEP file format.
3. Load a STEP File into GUI.
4. Type-Check Nodes, except for the expression STEP Node (expNode).
5. Manipulate multiple STEP programs simultaneously (as long as they do not share the same ID).
6. Execute STEP files, via connecting to SNBENCH.
7. Auto-generate Node options and enable the user to change them (using a right click menu).
8. Move and delete nodes.
9. Zoom-in and zoom-out (using mouse wheel)
10. Upload nodes dynamically (using Nodes.xml template).

6.3 Future Work

A possible direction for future work is to extend the current implementation to recognize SNAFU-formatted files. This can be done in two ways: (1) load the SNAFU file, convert it to STEP and then create the graph using the current implementation, (2) load the SNAFU file and directly convert it to a graphical representation.

Another possible direction is to improve the type-checker so that it can validate "expression" STEP nodes, which are not supported by the current implementation. Type-checking "expressions" is a pretty tough task and we intentionally removed it from the requirements specification, since it could end up being too complicated and we would diverge from the main goal of this project which is the graphical interface and not the type-checker.

Finally, another extension is to handle duplicate IDs in multiple STEP programs. This can be accomplished by modifying Michael Ocean's code accordingly.

References

[1] Azer Bestavros, Adam D. Brandley, Assaf J. Kfoury, and Michael J. Ocean. Snbench: A development and run-time platform for rapid deployment of sensor network applications. In *Proceedings of the IEEE International Workshop on Broadband Advanced Sensor Networks (Basenets 2005)*, Boston, MA, October 2005.

[2] Barbara Liskov and John Guttag. *Program Development in Java. Abstraction, Specification and Object-Oriented Design.* Pearson Education, 2005.

[3] Ian Somerville. *Software Engineering 7th Edition.* Pearson Education, 1996.