

Implementation of the Java SNePS 3 Building Box

SNeRG Technical Note 34

Jeffrey S. Fineberg
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
fineberg@buffalo.edu

May 4, 2003

Abstract

SNePS is a Knowledge Representation and Reasoning system that has been developed using the programming language Lisp. While this software is quite mature in its current state, there is a desire to develop SNePS into a graphical-based tool with a more ubiquitous programming language, in this case, Java. The main advantage of utilizing Java would be two-fold - SNePS could more easily work in multiple environments, and by using Java, this could aid in the development endeavor, as there are more programmers currently familiar with Java than Lisp. The purpose of this project is to continue the development of this Java version of SNePS, which was initiated in previous semesters. This paper discusses the overview of the project, recommended prerequisite knowledge for the development of the project, functionality that has been implemented and a final status report for continuing the development of this software.

1 Overview of the Java SNePS project

The intent of this project is to create a functional version of SNePS in Java. This version should take advantage of the features of Java, such as the built-in functionality of creating graphical user interfaces (natively in Java) and the ability to run on any platform that Java supports (Unix, Windows, etc.). Additionally it is expected that this version will not use a command line interface. Implementing SNePS in this manner benefits Knowledge Experts by allowing them to concentrate on creating knowledge representation structures without the concern of command line syntax [5].

2 Intended Audience

This paper is targeted primarily towards developers of this project, with the purpose being to provide an overview to increase their understanding of the project. Although the paper is primarily targeted towards developers, sections 10 and 11 have been provided for SNePS users regarding the operation of the Building Box and other components.

3 Recommended prerequisite knowledge for the developers working on the project

Due to the large scope of this project, it is recommended that new developers be familiar with areas that make up the system in order to gain an understanding of SNePS. Reading

several papers, viewing programming code and setting up a development environment as follows will help accomplish this. A more detail listing follows:

- Read several SNePS papers [1, 2, 3,4]
- Read the SNePS 2 User's Manual [9]
- Complete the SNePS 2 tutorial – “SNePS: An Interactive Approach”
- Read the SNePS 3 User's Manual [10]
- Read the paper “An Introduction to SNePS 3” [11]
- Read the document “Java and the Future of SNePS” [5]
- Review several Demo programs for SNePS 2 and SNePS 3.
- Review programming code for both the Lisp version and Java version of SNePS.
- Research various features of Lisp and Java that are unfamiliar when encountered in the code. [6,7,8]
- If it is desired to work with a remote development environment (at home or elsewhere), the developer may want to install Xemacs, Allegro Common Lisp and the Sun Java Development Kit version 1.4.

4 Functionality that has been implemented

A list of features and functionality is included to provide an overview of all the components that may be needed to complete the Java version (*refer to section 9 “Breadth First”¹ View of the Project*).

The main focus of development for this project has been to implement the Building Box (*see figure 1*). The purpose of this component is to enable the SNePS user to create pieces of SNePS networks. This is done by the user selecting a case frame from a list of valid choices. Once selected, a molecular node representing the selected case frame would appear wherever a click is performed. The user would then modify the labels and other information as desired. Although not currently implemented, when the information is correct, the user would then “Assert” or “Build” the nodes that were created – these nodes would then be a part of the SNePS network (*for details, refer to section 10 “Building Box user notes”*).

The following list is a detailed account of the implementation of functionality that was implemented in the Building Box:

- The Building Box component was implemented to allow the creation of nodes and directed edges with labels for the purposes of building networks (*see Figure 1*).

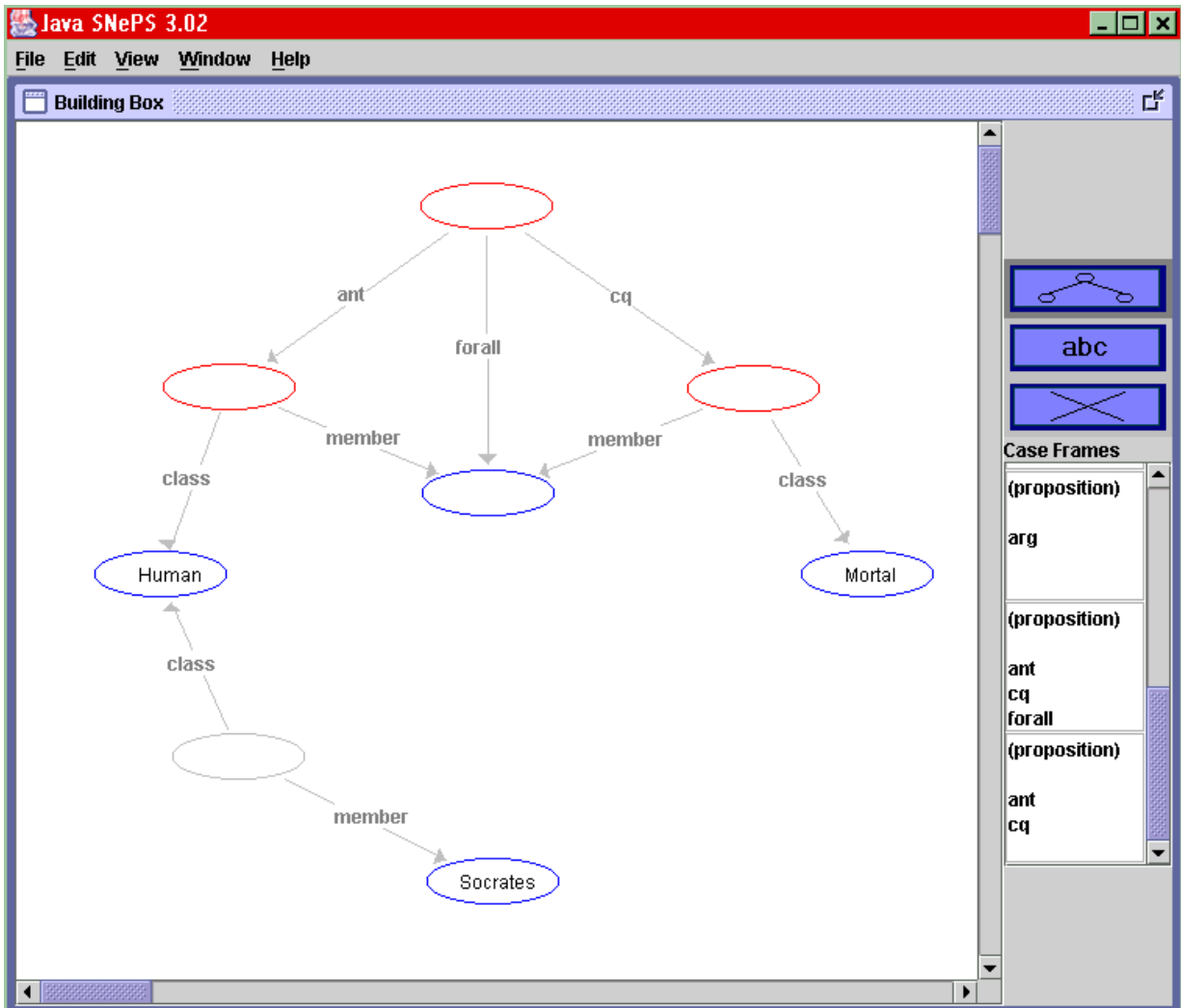
Features include:

- o User-friendly interface, containing a “toolbox style” method of editing.
- o Creation of molecular nodes, utilizing case frames for creating basic structures that are populated by the user.
- o Labeling of base nodes.

¹ A Term used by Dr. Shapiro to describe the scope of the project list.

- Duplication of wires.
- Deletion of wires, with relation minimums being enforced.
- Deletion of molecular nodes.
- Merging of nodes, including the management of semantic classes pertaining to the nodes. This management is also performed when an edge is removed from a node.
- Moving of nodes is done with a mouse, with connected edges following.
- Nodes and edges are implemented as data structures, for later integration with the JavaSNePS network data structures (*refer to Section 5 “Informal overview of class relationships in Java SNePS”*)
- The following source files have been modified: GUI.java, GraphNode.java, S3_CaseFrame.java and BuildingBoxFrame.java (refer to Javadoc documentation for details).
- The following source file was created to support edge functionality: GraphEdge.java (refer to Javadoc documentation for details).
- The modified source code is in the following directories. Package layout is described in SNeRG Technical Note 31 [5].
 - /projects/snwiz/Javasneps3/Sneps3/sneps3/world/gui
 - /projects/snwiz/Javasneps3/Sneps3/sneps3/corecode
- Javadoc style comments were added to all source code that was modified or created, including GUI.java, which was not using Javadoc style comments previously.
- All source code Javadoc documentation was regenerated, using the shell script ‘makedocs’.
- README file modification: instructions for using the Building Box features were created for use in on-line help. Instructions were also added for developer’s use.
- On-line help is now available in JavaSNePS by selecting the menu option ‘Help’, then ‘User Manual’. The help information is not hard-coded, but is extracted from a README file (previously mentioned), which is located in the local directory of the Java application.
- New Jar utilities were implemented with two new shell scripts:
 - makejar_executable – creates a jar file (named jsnepsjar_executable) containing all .class files.
 - makejar_install – creates a jar file (named jsnepsjar_install) containing the SNePS jar executable file (jsnepsjar_executable), the README file and the images directory which contains GIF images for buttons. This jar file contains everything needed to run the program on a system running JDK1.4. Refer to section 8 for details of installation.
 - makejar_all_files – creates a jar file (named jsnepsjar_all_files) containing all files (source, executable, text, etc.) from the current directory and all subdirectories. This is useful for transporting the entire system for a developer to work with (keeping in mind that CVS checkout should be considered if modifications are being made) [5].
- Changed the version number on the title bar from 3.0 to 3.01 (*December 2002*), and most recently (*May 2003*) to 3.02 for differentiation purposes.

Figure 1. Example of a JavaSNePS graphical representation using the Building Box (from the SNeRG web site at <http://www.cse.buffalo.edu/sneps/>)



5 Informal overview of class relationships in Java SNePS

The classes used in Java SNePS are primarily used for either one of two purposes - data structures that comprise the SNePS network or graphical components used for the presentation and manipulation of the SNePS network.

Graphical Components

These are composed of several “frame” classes - a group of classes that comprise a `JInternalFrame` within the main `Frame` class, `GUI.java`. These components are used for GUI operations and integration with the SNePS network. The naming convention is consistent with the type of data structure it supports.

The Frame classes are:

BuildingBoxFrame.java CaseFramePanel.java RelationFrame.java
 CaseFrameFrame.java ContextFrame.java NetworkFrame.java
 SemanticClassFrame.java

Data structure components

The following classes are used for implementing the data structures of the SNePS network. *The + symbol is used as a shorthand for 'and'.*

Class name	Composed of
Context	(String <u>name</u>) + (NodeSet <u>hyps</u>) + (NodeSet <u>ders</u>)
ContextSet	(Hashtable of Contexts) - key is Context <u>name</u> (unique)
SemanticClass	(String <u>name</u>) + (SemanticClass <u>parent</u>) + (SemanticClassSet <u>children</u>) + (NodeSet <u>nodes</u>) + (RelationSet <u>relations</u>) + (CaseFrameSet <u>caseframes</u>) + (Color)
SemanticClassSet	(Hashtable of SemanticClasses) - key is SemanticClass <u>name</u> (unique).
Relation	(String <u>name</u>) + (SemanticClass <u>type</u>) + (int <u>adjust</u>) + (int <u>limit</u>) + (Path <u>path</u>) + (Nodeset <u>nodes</u>) + (CaseFrameSet <u>caseframes</u>) + (int <u>id</u>) + (boolean <u>lock</u>)
RelationSet	(Hashtable of Relations) - key is Relation name (unique)
CaseFrame	(SemanticClass <u>semanticClass</u>) + (RelationSet <u>relations</u>) + (NodeSet <u>nodes</u>) + (String <u>hashValue</u>) (boolean <u>lock</u>)
CaseFrameSet	(Hashtable of CaseFrames) - key is a concatenation of Case frame data.
Node	(String <u>name</u>) + (SemanticClass <u>semClass</u>) + (int <u>id</u>) + (ContextSet <u>assertedIn</u>) + (CableSet <u>upCables</u>) + (Object <u>slot1</u>) + (Object <u>slot2</u>) + (Object <u>slot3</u>) + (boolean <u>lock</u>)
NodeSet	(Hashtable of Nodes) - key is Node name (unique)
BaseNode	<i>Extends Node</i> (nothing additional declared).
MolecularNode	<i>Extends Node.</i> (CableSet <u>cableSet</u>) + (CaseFrame <u>caseFrame</u>) + (boolean <u>lock</u>)
Cable	(Node <u>root</u>) + (Relation <u>relation</u>) + (NodeSet <u>nodeset</u>)
CableSet	(Hashtable of Cables) - key is the relation name associated with each Cable (unique).
Network	(NodeSet <u>nodes</u>) + (ContextSet <u>contexts</u>) + (CaseFrameSet <u>caseFrames</u>) + (RelationSet <u>relations</u>) + (SemanticClassSet <u>semClasses</u>) + (Context <u>defaultContext</u>)
Path	(int <u>pathtype</u>) + (Relation <u>base</u>) + (Path <u>firstPath</u> , <u>restPath</u>) + (boolean <u>converse</u>) + (boolean <u>lock</u>).

6 Location of modified code, scripts and documentation

The standard location of the source code, documentation, jar files and related scripts is:

```
/projects/snwiz/Javasneps3/Sneps3/sneps3
```

Important note: when modification of code is performed, this work must be done in a developer's environment, not directly in the above directory. This will assure that a stable version of the software is always available. Once work is completed and this directory is updated, the CVS repository must also be updated for version control [5].

7 Executing the code on the CSE Unix systems

This can be done as follows:

```
> cd /projects/snwiz/Javasneps3
> jsneps
```

Alternatively, the program can be executed using the jar file in the following directory:

```
> cd /projects/snwiz/Javasneps3/Sneps3/sneps3
> java -jar jsnepsjar_executable
```

8 Installing and executing the code in other environments

Testing of this version has been performed in the following environments: Windows ME, Windows 2000, Solaris and Redhat Linux.

To execute this program on various machines, FTP (in binary mode) a copy of the jar file 'jsnepsjar_install' in order to run this locally on your machine. Once it is in a directory of your choosing (assuming you have the JDK 1.4 set up in your path), do the following:

```
> jar xvf jsnepsjar_install (unpacks the necessary files and images directory)
> java -jar jsnepsjar_executable (executes the program)
```

9 "Breadth First" View of the Project

The details below are a general overview of outstanding tasks necessary to complete the development of the Java version of SNePS 3. This is an extensive list, useful for discovering the scope of the various work involved and should help in organizing and prioritizing this large scale project. This list should be updated as new information is obtained.

Building Box component

- Enforce uniqueness of node labels. When adding labels to nodes, there are 3 situations to be accommodated:
 - Node doesn't exist
 - Node exists in the network, but not in the Building box
 - Node exists in the Building Box, but not in the network
- Font size adjustment for labels.
- Undo functionality for an operation such as create and delete.
- Node size adjustment (based upon amount of text).
- A 'commit' button should populate the SNePS network. All committed propositions must be constrained to be valid case frames.
- How to graphically handle more than 1 edge between nodes, with unique relations {consider creating a label to act as a node, in order to 'pull' edges apart for readability}.
- Duplicate several nodes and edges (by highlighting a group of selected nodes).

Network component

- Modify the existing Network component to be able to display a selected portion of the network.
- Create a method of displaying the network automatically, that a user could then override by moving nodes around to suit their viewing.
- Node point positions (location on screen) should be stored with the GUI data structures, not as part of the knowledge structure, to allow abstraction.

General

- Resize / reposition windows upon startup for easier use. Allow saving of User selected start up windows position.
- Modification of the code to run as an applet also, as indicated in code comments.
- Continue developing data structures for representing the network, if necessary (*refer to 5 "Informal overview of class relationships in Java SNePS"*)
- Network window implementation, possibly based directly upon the Building Box component.
- Node exists in the Building Box, but not in the network
- Assert and Build functionality (utilizing Building Box and Network GUI).
- Dump / describe functionality (allNodes).
- View functionality of assertions / ders / hyps.
- Find functionality – some sort of query window? *Use building box to create structure for querying?*
- Functions on sets of nodes (union, intersection, difference, etc.).

- SNIP – algorithms and structures needed
 - Node-based
 - Path-based
 - Wire-Based
 - Subsumption
- Resetting the network.
- Saving and loading of networks (i.e. outnet / innet). *Using serializable should enable this, unless data is stored similar to Lisp SNePS.*
- Export of networks information to the DOT format for display purposes using software such as Graphvis.
 - (refer to <http://www.research.att.com/sw/tools/graphviz/>).
- Demo JavaSNePS functionality (*possibly use <http://abbot.sourceforge.net> The Abbot framework provides automated event generation and validation of Java GUI components. Also refer to the Java Robot class, as it also contains automated functionality*).
- Snepslog equivalent functionality.
- Enhanced help utility with index and search functionality.

Documentation / Testing


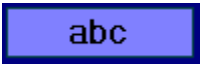

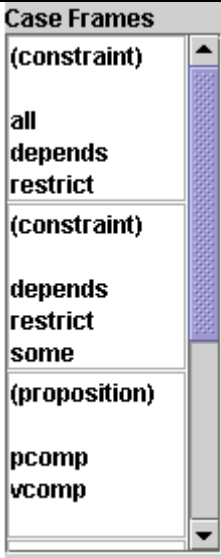
- Demo scripts for testing functionality – *represent SNePS 3 demos, in the order in which they appear.*
- User’s manual updating as necessary.
- Continue additional detailed documentation in the code where appropriate.
- Status report overview upon the semester’s completion.

10 Building Box User Notes

The Building Box is used for creating pieces of SNePS networks. Using a graphical interface (similar to a paint program or Visio), molecular nodes are created from case frames initially. Nodes (molecular and base) can be connected to form larger segments of networks. The program utilizes constraints (referred to as rules below) to assure as best as possible that any configuration of nodes and edges comprises a “legal” SNePS network.

Description (see Figure 1 on page 4 for a sample screen)

The Building Box consists of 4 controls, which are described below:

Display object	Description
	<p>Case frame creation - once selected, a case frame is created wherever the user clicks on the screen.</p> <p>Wire duplication - additional wires can be created in this mode by left-clicking an edge label.</p> <p>Node moving – any node can be moved by clicking and dragging it.</p> <p>Node consolidation – nodes can be merged together. See the description below for rules of node consolidation.</p> <p>Node and Edge deletion – done by right-clicking a node or label.</p>
	<p>Text box – once selected, the user can type text into nodes. This is not possible for edges as these are defined as relations for a given case frame.</p>
	<p>Delete – this can be used for deleting nodes and wires, provided that the deletion doesn't result in an illegal network. This functionality can alternatively be done using the right-click of the case frame function. See the description below for rules of node and wire deletion.</p>
	<p>Case frame selection – this scrolling window is used in conjunction with the Case frame creation function. Once a case frame is selected, when an area is clicked on the Building Box graphical area, a case frame is rendered in that area, using the information in the case frame. Minimally, one wire is created for every relation in a selected case frame. However, if a relation has a limit other than zero, the number of wires created for that relation is the limit value. Also note that, when an edge is deleted, validation is performed to assure that the minimum number of relations exist, otherwise a warning will alert the user and deny the deletion.</p>

Starting the program – refer to sections 7 and 8 for running the program in your environment.

Rules of node consolidation – the following is the various conditions in which the merging of nodes is allowed (condition 1, 2, and 3 must be met):

1. The labels are the same

- or -

One of them is blank

- AND -

2. Both nodes are base nodes

- or -

One node is molecular

- AND -

3. The nodes have the same semantic class or the semantic class of one is a descendant of the semantic class of the other. When nodes merge, the resulting merged node obtains the lower of the two semantic classes.

Rules of wire deletion – When a wire is deleted and the base node it points to has no other wires pointing to it, then both the wire and node are deleted. When a wire is deleted that points to a base node and other wires are still pointed to it, the semantic class of that node is recalculated and revised based upon the semantic class of all the relations pointing to it. Similar to node merging, the resulting node obtains the lowest of the semantic classes.

When a wire is deleted that points to a molecular node and other wires are still pointed to it, the semantic class of that node is recalculated and revised based upon the semantic class of all the relations pointing to it and the original semantic class of the case frame for which the molecular node represents. Similar to node merging, the resulting node obtains the lowest of the semantic classes.

Rules of Molecular node deletion – when a molecular node is deleted, all of its edges are deleted, using the “rules of wire deletion” (*noted above*). However, a node which is pointed at by the molecular node’s edges is only deleted if there are no other wires pointing to it.

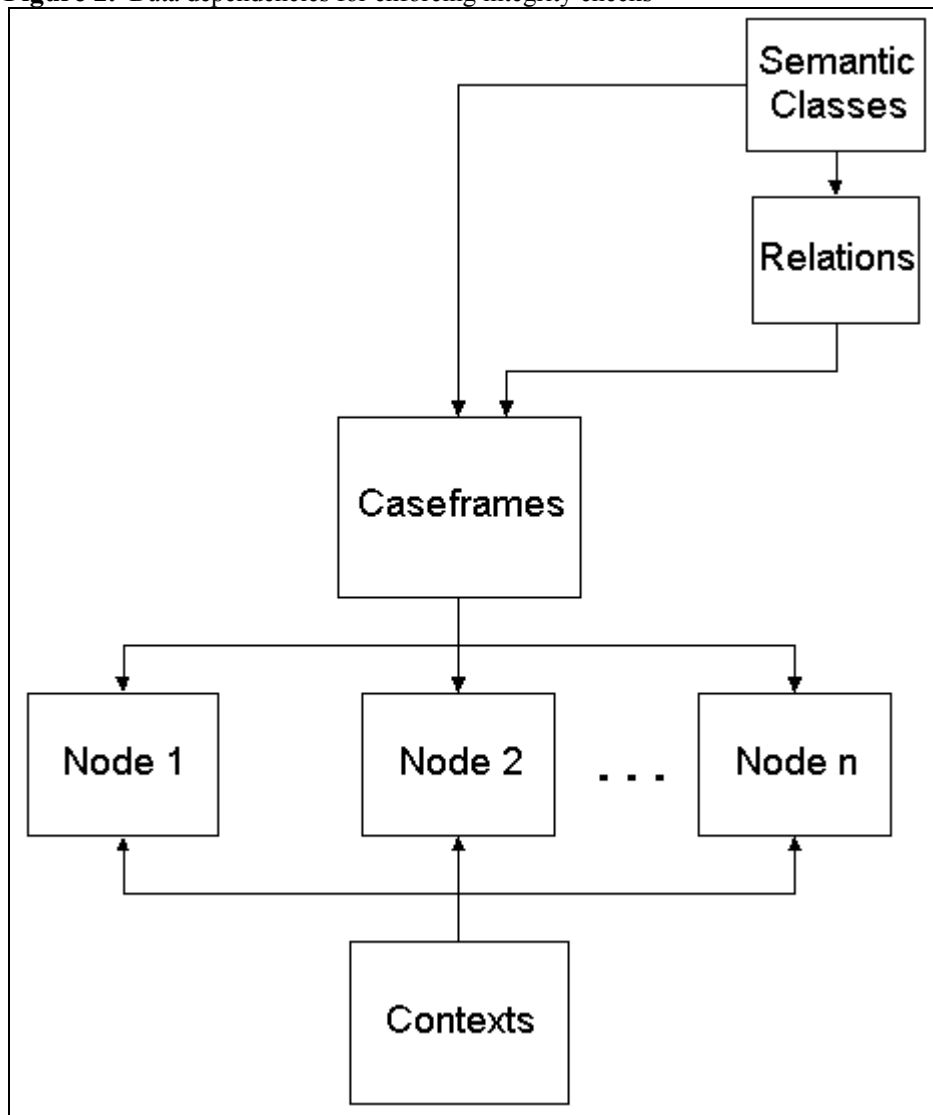
How to perform an ‘undo’ operation using edge duplicate and remove – if a user merges two nodes together and later realizes that it was performed by mistake, there is no need to delete and create new molecular nodes. Simply duplicate one of the wires and then delete the original wire that connects them. *A future release may contain an undo button for this situation.*

11 JavaSNePS Overview for Users

The following is an overview of the Java version of SNePS. Using the similar principle as that of the Lisp version of SNePS 3², the following constraints are enforced during program runtime (*illustrated in Figure 2 below*):

- Relations must contain valid semantic classes.
- Case frames must contain valid relations and semantic classes.
- Nodes must consist of valid case frames.
- Nodes must be asserted in a valid context (*not yet implemented*).

Figure 2. Data dependencies for enforcing integrity checks



² Further discussion of the components can be found in documentation for the Lisp version - “An Introduction to SNePS3” [11] and “SNePS 3 User Manual” [10].

Description of the user components of Java SNePS – the following is an overview of each component used in the process of creating networks (see Figure 3):

Semantic class component – allows the user to create semantic classes using a tree structure, representing a hierarchy relationship. Each semantic class may have any number of descendants. The relationship of semantic classes is used for calculating semantic classes of merged nodes. Colors are assigned to each semantic class for improving the readability of the diagrams.

Relations component – used for the creation of relations which represent edges connecting molecular nodes to other nodes. The user must select a semantic class from a list of valid choices. The adjust parameter “specifies how wire-based inference treats instances of this relation” [10]. Limit is “the minimal size of a nodeset that can be paired with this relation in a cable” [10]. This value is used by the building box during the creation of molecular nodes and also for enforcing the minimum number of relations during wire deletions.

Case frame component – used for creating case frames, which are then used by the building box for creating molecular nodes (*or “instances of case frames [11]”*). When creating a case frame, the user must select from a list of valid semantic classes and also valid relations.

Building Box component – used for creating pieces of SNePS networks. Using a graphical interface (similar to a paint program or Visio), molecular nodes are created from case frames initially. Nodes (molecular and base) can be connected to form larger segments of networks. The program utilizes constraints (referred to as rules below) to assure as best as possible that any configuration of nodes and edges comprises a “legal” SNePS network.

Contexts component – used for creating and deleting contexts. Also included is the ability to designate a particular context as the default context when asserting a node. *This is not fully implemented, as it is dependant upon the building box and network component development (See figure 4 for an example).*

Network component – used for displaying portions of the network, consisting of any number of assertions. This component has not been implemented to a detailed degree at this time.

Figure 3. The JavaSNePS application screen as it appears upon startup (*context and network components not shown*)

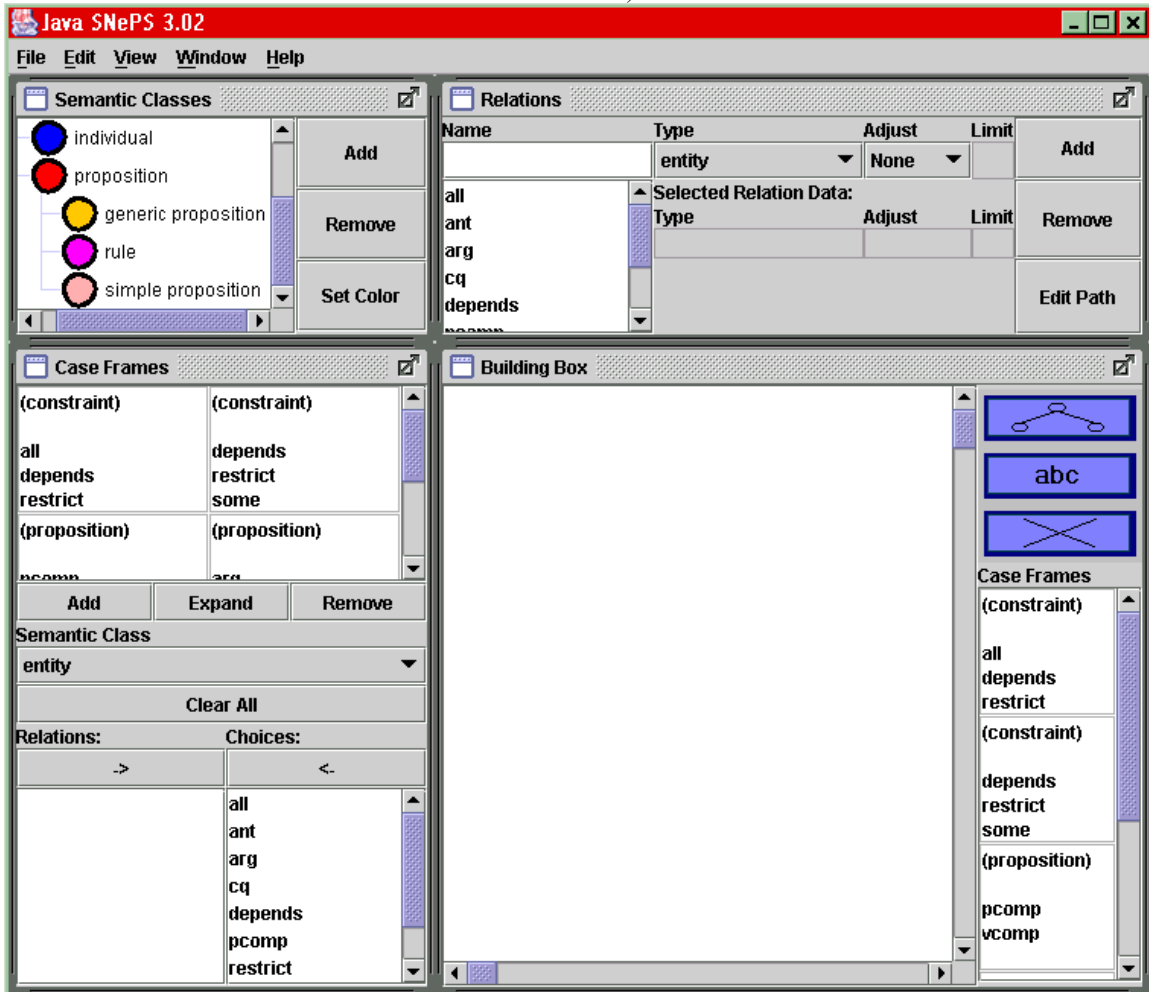
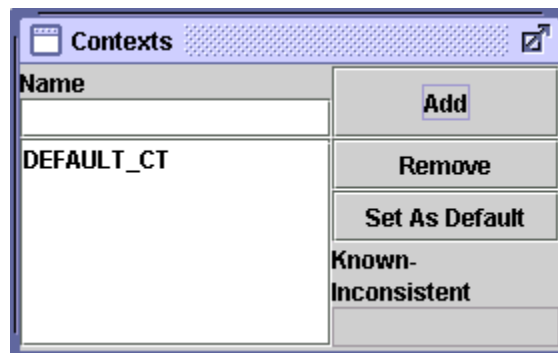


Figure 4. The JavaSNePS Contexts component display



References

- [1] Shapiro, S. C. (1991) Cables, Paths, and “Subconscious” Reasoning in Propositional Semantic Networks. In J. Sowa (editor) *Principles of Semantic Networks: Explorations in the Representation of Knowledge* (Los Altos, CA: Morgan Kaufmann) pp. 137-156.
- [2] Shapiro, S. C. and Rapaport, W. J. SNePS (1987) Considered as a Fully Intensional Propositional Semantic Network. In N. Cercone and G. McCalla (editors) *The Knowledge Frontier* (New York: Springer-Verlag) pp. 263-315.
- [3] Shapiro, S. C. (1989) The CASSIE Projects: An Approach to Natural Language Competence. In J. Martins and E. Morgado (editors) *Lecture Notes in Artificial Intelligence* (Berlin Heidelberg: Springer-Verlag) pp. 362-380.
- [4] Shapiro, S. C. and Rapaport, W. J. (1992) The SNePS Family. *Computers and Mathematics with Applications*, 23(2-5): 243-275. Reprinted in F. Lehmann (editor) *Semantic Networks in Artificial Intelligence* (Oxford: Pergamon 1992) pp. 243-275.
- [5] Petre, A. (2001) Java and the Future of SNePS. SNeRG Technical Note 31 (Buffalo, NY: SUNY Buffalo Department of Computer Science)
- [6] Shapiro, S. C. (1992) *Common Lisp – An Interactive Approach*. (New York: Oxford)
- [7] Steele, G. L. (1990) *Common Lisp the Language*, 2nd Edition . (Woburn)
- [8] Deitel, H. M. and Deitel, P. J. (1999) *Java – How to Program*. (New Jersey)
- [9] Shapiro, S. C. and The SNePS Implementation Group (2002) *SNePS 2.6 User’s Manual* (Buffalo, NY: SUNY Buffalo Department of Computer Science)
- [10] Shapiro, S. C. and The SNePS Implementation Group (2002) *SNePS 3 User’s Manual* (Buffalo, NY: SUNY Buffalo Department of Computer Science)
- [11] Shapiro, S. C. (2000) An Introduction to SNePS 3. In Bernhard, Ganter and Guy W. Mineau (editors) *Conceptual Structures: Logical, Linguistic and Computational Issues. Lecture Notes in Artificial Intelligence* (Berlin: Springer-Verlag) pp. 510-524.