Java and the Future of SNePS
SneRG Technical Note 31

Anthony Petre
Department of Computer Science and Engineering
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
November 16, 2001

1 Introduction

This paper explains the initial design of the Java version of SNePS 3. It also covers the ideas that came out of the design project, and makes many references to potential areas of future works with the system.

This paper assumes the reader has an understanding of previous versions of SNePS and a familiarity with the Java programming language.

2 Goals

Java SNePS 3 was designed with a number of goals in mind:

- The system should provide a useful research tool utilizing past and current theories of the SNePS research group.
- The system should be built using Java and standard Java programming techniques to make the software more accessible to the modern research body.
- The code should be written simply and clearly as possible, since many future users will need to understand how the code works. This includes writing well-documented code.
- As the initial project was not expected to complete the entire system, there should be a strong framework for future projects to build on.
- The code should be highly adaptable, providing not only for current projects but also trying to anticipate future ones.

This paper will try to include the reasoning behind many of the design decisions. If some aren't covered, one of the above considerations is likely behind it.

3 Package Organization

The code for Java SNePS 3 has been divided between several packages, divided logically by its role in the system. The initial packages are:

sneps3.classes

Contains the public interfaces for common SNePS objects (e.g. Nodes). These are meant to be very simple, mainly just what would be needed for display of network information. Modification of the objects is generally disallowed, with the main exceptions being anything that is only display-related (like the color scheme for semantic classes) and the Network class (which is meant to give the same abilities as SNePSUL).

This package is about 95% complete, missing mainly the code for paths.

sneps3.corecode

This package contains more complete interfaces for the common SNePS objects of the sneps3.classes package (extending them even). This package also contains a default implementation of each of these interfaces (the class names being prefaced with "S3_").

The particular implementation used at any one time is set in the SnepsFactory class, which essentially provides the correct constructors for the network to use.

This may seem overly complicated, but it makes alternate implementations easy. If a new implementation of a particular class is created (maybe with efficiency improvements) then only the SnepsFactory class need be changed to use the new implementation (provided the new version implements the correct interface).

This package is about 95% complete, missing mainly the code for paths.

sneps3.data

This package is currently unused, it's kept more as a storage dump for any data the system might want to store (lexicons, statistics, etc).

sneps3.event

This package has the code for the event model. It was hoped to make all network changes cause events that could be caught by other parts of the system and acted on, but it was never fully implemented. It is intended as an extension of the existing Java event model.

This package is about 75% complete. It needs a little more planning and work.

sneps3.exception

This package has the code for the exception model. It is intended as an extension of the existing Java exception model.

This package is about 75% complete. It needs a little more planning and work.

sneps3.mind (Manager of Inference and Network Domains)

This package holds code for the construction of SNePS "minds". A mind brings all the elements of the system together. See the system diagram in appendix A and the description in section 5 for a better idea how it relates to the system.

sneps3.snebr (SNePS Belief Revision)

This package is for code to manage belief revision. Maintenance of removed beliefs, algorithms for dealing with inconsistencies, source information managers, and any other such related parts would go in this package.

This package is not yet implemented.

spens3.snere

This package relates to the SNePS 2 SNeRE component, but is outdated in the new hierarchy and should be removed.

sneps3.snip (SNePS Inference Package)

This package is for code to perform reasoning and inference. Node-based, path-based, abductive, and all other types of reasoning stem from this package.

This package has not yet been implemented.

sneps3.spine (SNePS Perceptor Interface and Neural Effectors)

This package is for connection and translation code to formalize incoming data into nodes that the network can use, and to reformat node information for use by outside sources.

The code in this package might simply take incoming natural language and represent it with a node structure. It might similarly translate data from an image sensor into a node structure.

This package would also handle communication in the other direction. A command from the network to carry out a plan represented by a node structure would then be translated into a series of instructions in the hardware's language.

This package handles what was previously part of SNeRE, plus additions.

sneps3.world

This package is for code that isn't directly a part of the SNePS system. GUIs and simulations would fit in this category.

Most systems added here should be in their own sub-package. For example, the default GUI is being built in sneps3.world.gui.

4 Coding Style

The existing code has been documented in the javadoc standard. It is suggested that future programmers learn and use this system.

5 MIND

The sneps3.mind package provides code for combining different sections of the SNePS system into an agent. When building a mind the programmer needs to decide what elements will be needed, it determines the *kind* of agent created.

One or more networks will be a part of any mind. There may also be inference components (from sneps3.snip) or belief management components (from sneps3.snebr). These three sections are normally built by the same programmer or group, intended to work closely together.

Components reaching outside the mind (from sneps3.spine) are managed by the mind code. The mind coordinates communication between these exterior parts and the internal systems. For example, natural language input might come into the mind. The mind may then query the inference package with the incoming data to determine source information and later pass that information to the belief revision system.

As a manager, the mind would be a good area to place resource management code as well. For example, it might make decisions on how much processing time or memory to devote to a given task. It might well rely on the inference package to help make such decisions.

Given it's position as the last step before direct connection to the network, the mind is also an ideal location to handle any security protections for the inner components.

There is a default mind implementation called OpenMind, which is meant to give transparent access to the interior components. This is normally sufficient when security is not an issue and connections between system parts are simple (such as doing research work all on one machine).

6 SPINE

The spine code is meant to free the rest of the system of the hassle of connecting to exterior components. KIF-to-SNePS, LKB, and other translators would go here.

Code for turning device input into SNePS forms would also be placed in this package. The network and inference package might design a micro-program of multiple steps that can be sent to a device, and later run with a single command. Also, if a device is modified, upgraded, or replaced it may often be possible to simply alter the code in the spine package to handle all the changes.

If the agent needs to communicate with other programs, even running on different computers, the spine package should have code for this as well.

A default spine connection called TelePort (a telepathic port) is provided, which gives direct access to an OpenMind (which thereby gives direct access to the network therein). The default GUI uses this connection, even though realistically it could simply connect to an OpenMind directly.

7 Style Standards:

- Use javadoc commenting style.
- Use standard Java naming conventions (initial upper-case class names, lower-case field and method names, all caps for constants, etc.
- SNePS 3 objects should implement the Serializable and Cloneable interfaces.
- SNePS 3 object interfaces should declare the clone method to be public.
- To maximize structure sharing, clone methods should all do shallow copying (and thus a shallow copy clone should be assumed when using it).
- Make parameters **final** (constant) whenever possible.
- binary operators between two objects of the same class should have direction indication in their name and 2nd parameter called rhs (right-hand-side) to clarify direction of operation
- in absence of more informational name, use abbreviations for parameter based on class: sc for semantic class, c for cable, n for node, ns for node set, etc.
- All objects should have a toString method, and a version of toString that takes as a parameter a level of indentation. The default toString could simply call the parameter version with a value of 0, but a simpler output is better since many Java graphical renderers use the toString method to create simple labels for objects, and listing all the information creates too large a label.
- Implementation files should list fields first, then methods not declared in interfaces, and finally methods declared in the interfaces (which do not require javadoc headers, but will inherit them from the interface).
- Fields should be **protected** where possible (controls errant access but allows easy creation of new implementations by extending existing ones)
- SnepsExceptions should generally NOT be thrown in the lower corecode classes (SemanticClass, Relation, etc) since methods there may be needed in static initializers (which can't use methods that throw exceptions, even if they get caught).
- Network is the first layer of the system that should really be throwing exceptions.

8 Getting and Running the Code

To get the current working version of the code, you need to create a new folder, change to it and type::

```
cp /projects/snwiz/Javasneps3/sneps3.jar .
unzip sneps3.jar
chmod –R u+x *
```

This will unzip the files into a directory called sneps3[1]. Execute permissions don't get saved it seems, so the last command gives you access to the sub directories. This makes ALL the files executable though, so you may wish to go through afterwards and remove the unnecessary permissions.

[1] This has been done. The directory containing all the code is /projects/snwiz/Javasneps3/Sneps3. To run the Java version of SNePS 3, execute /projects/snwiz/Javasneps3/Sneps3/sneps3/jsneps3.
[Stuart C. Shapiro, 6/7/02]

In the sneps3 directory there is a README file which gives you information about the scripts provided for compiling, running, and managing the code.

9 Accessing and Using the CVS Code Repository

Before you can access and use the repository, you need to get read and write access to the /projects/snwiz/Javasneps3/ directory. The easiest way is to get added to the cse-jsne group, which already has these permissions.

Once you have these permissions, you need to set that directory to your CVSROOT environment variable each time you log on and wish to use the repository. The easiest way to do this is to add the following line to your .login file:

setenv CVSROOT /projects/snwiz/Javasneps3

To begin doing work with the repository without logging back in, you can just type:

source .login

Now create a directory to store your version of the code and cd into it. You need to checkout the files and the CVS information so that the system can keep the files updated. To do this type:

cvs checkout sneps3

This will copy all the code to your local directory. If you want to add a file or directory to the repository, first create the file or directory. Then type the following:

cvs add filename

Without this command, the file will still exist in your directory, but won't be a part of the repository, and thus won't be managed by the CVS system. If you want to remove a file or directory or file from the repository, type the following:

cvs remove filename

If you are removing a file, you will have to delete the file before it can be removed. If there are other people working with the repository, they may make changes to the files and save them to the repository. To get these changes type the following:

cvs update

This command will update all the files in the directory you are in and any sub directories. For more information on the CVS system, see the man pages on cvs.

Appendix A – System Diagram